MPEG-2 视频解码器的 FPGA 设计

摘　要

MPEG-2 是 MPEG 组织在 1994 年为了高级工业标准的图象质量以及更高的传输率所提出的视频编码标准，其优秀性使之成为过去十年应用最为广泛的标准，也是未来十年影响力最为广泛的标准之一。

本文以 MPEG-2 视频标准为研究内容，建立系统级设计方案，设计 FPGA 原型芯片，并在 FPGA 系统中验证视频解码芯片的功能。最后在 0.18 微米工艺下实现 ASIC 的前端设计。完成的主要工作包括以下几个方面：

1. 完成解码系统的体系结构的设计，采用了自顶而下的设计方法，实现系统的功能单元的划分；根据其视频解码的特点，确定解码器的控制方式；把视频数据分文帧内数据和帧间数据，实现两种数据的并行解码。
2. 实现了具体模块的设计：根据本文研究的要求，在比特流格式器模块设计中提出了特有的解码方式；在可变长模块中的变长数据解码采用组合逻辑外加查找表的方式实现，大大减少了变长数据解码的时间；IQ、IDCT 模块采用流水的设计方法，减少数据计算的时间；运动补偿模块，针对模块数据运算量大和访问帧存储器频繁的特点，采用四个插值单元同时处理，增加像素缓冲器，充分利用并行性结构等方法来加快运动补偿速度。
3. 根据视频解码的参考软件，通过解码系统的仿真结果和软件结果的比较来验证模块的功能正确性。最后用 FPGA 开发板实现了解码系统的原型芯片验证，取得了良好的解码效果。

整个设计采用 Verilog HDL 语言描述，通过了现场可编程门阵列(FPGA)的原型验证，并采用 SIMC 0.18μm 工艺单元库完成了该电路的逻辑综合。经过实际视频码流测试，本文设计可以达到 MPEG-2 视频主类主级的实时解码的技术要求。

关键词：　MPEG-2，视频解码，硬件设计　原型验证

FPGA Design of MPEG-2 Video Decoder

ABSTRACT

MPEG-2 standard was established for higher industry standard of picture and higher transport rates by Moving Picture Expert Group. The standard is so excellent that it becomes the most influential standard since ten years ago, and is still one of the most important compress standard nowadays .

The research content of thesis is the standard of MPEG-2 video. Firstly, the total system architecture of MPEG-2 video decoder was introduced; then design of the FPGA prototype and its verification were achieved; at last, in the SMIC 0.18μm standard cell CMOS technology, the decoder is implemented targeting ASIC. The contributions of the essay are summarized as following:

1. The architecture design of MPEG-2 video decoder is implemented. With Top-Down method, the decoder system was divided into some sub-modules; based on the characteristic of video decoding, define the control method of system; divide the video datas into intra-data and inter-data, then the two kinds of data was decoded by parallel algorithm.

2. the design of sub-modules: according to the system design ,the parser module was designed by a unique method; In the VLD module, taking the advantages of both combinational logic and LUT, accomplished the decoder of variable data, reduce the decoding time obviously; IQ,IDCT module was designed by pipeline method, also reduce the time of calculating heavily; For the motion compensation is the most intensive part of accessing memory and has high throughput, several methods were adopted to improve memory access efficiency, including four pixel interpolator, pixel buffer augmentation, and parallel architecture, which lead to good speedup of motion compensation.

3. Based on the software reference program, functional verification is carried out by comparing the simulation results with the output results of software. At last, the FPGA prototype is implementated based on the developing board EP2S180 from Altera. FPGA prototype based verification showed that the design gets satisfactory decoding effects.

The design was described with Verilog HDL. It has been implemented by Field Programmable Gate Array (FPGA). Synthesis was also fulfilled with Synopsys Design compiler, based on SMIC 0.18μm standard cell CMOS technology. Demo experiment with real video stream shows that the design can meet the real-time decoding requirements of the MPEG-2 MP@ML video sequence.

Key words: MPEG-2, Video decoding, Hardware Design, FPGA prototype verification

插图清单

表格清单

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___合肥工业大学___或其他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。

学位论文作者签名：[signature]　　签字日期：09 年 4 月 20 日

学位论文版权使用授权书

本学位论文作者完全了解_合肥工业大学_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_合肥工业大学_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适用本授权书)

学位论文作者签名：[signature]　　导师签名：[signature]

签字日期：09 年 4 月 20 日　　签字日期：09 年 4 月 20 日

学位论文作者毕业后去向：

工作单位：　　电话：

通讯地址：　　邮编：

致谢

本论文在选题和研究过程中受到高明伦教授的悉心指导。高老师不仅学识渊博、治学严谨，而且待人诚恳，平易近人。他对科学严谨认真的态度和高尚的品德，是我学习的榜样。这两年多来从高老师那里学到的知识必将使我终身受益。衷心感谢导师的培养、支持和教诲！两年多来，老师们不仅在学业上给予我精心的指导，而且在思想、生活方面给予我无微不至的关怀，在此谨向他们致以诚挚的谢意和崇高的敬意！

感谢张多利老师在论文选题、结构确定、审核修改方面给我的帮助和细心指导，这些促使我论文一步步走向完善。

感谢张多利老师、杜高明老师、耿罗锋博士和侯宁博士在项目中的热情指导和帮助！

感谢宋宇鲲老师在小论文写作中的耐心指导与帮助！

感谢 NOC 项目组马亮、付强、翟元杰、杜福慧、王白露、尹凯、程贤文、吴腊狗、于亚轩所给予的帮助！

感谢邓红辉老师、林微老师、贾靖华老师在学习和生活上给予的支持和帮助！

感谢微电子设计研究所全体成员，正是由于你们的帮助和支持，我才能克服一切困难和疑惑，直至本文的顺利完成。

特别感谢我最亲爱的家人，他们对我的深厚爱意和默默支持是我学习和工作的动力。你们对我的养育之恩，我终生无以回报！

张云

2009 年 4 月

第一章 绪论

1.1 MPEG 标准概述

随着信息时代的到来，特别是在信号处理技术发展起来的时候，图像处理的技术逐渐数字化，由于许多新的理论和方法的引入，出现了一系列的图像压缩算法[1]，因此在二十世纪九十年代左右，这些经典的视频压缩算法逐渐形成一系列的国际标准体系[2]，比如 H.26X 系列以及 MPEG 系列等。

其中，MPEG 作为重要的一种编码方式[2]，得到了最为广泛的应用。MPEG 就是动态图像专家组（Motion Pictures Experts Group）的英文缩写，始建于 1988 年，是专门制定多媒体领域内国际标准的一个组织。由全世界大约 300 名多媒体技术专家组成。制定的标准包括 MPEG 视频、MPEG 音频和 MPEG 系统（视音频同步）三个部分。他们成功地将声音和影像的记录脱离了传统的模拟方式，建立了 ISO/IEC1172 压缩编码标准，并制定出 MPEG-X 格式，令视听传播方面进入了数码化时代。因此，大家现时泛指的 MPEG-X 版本，就是由 ISO(International Organization for Standardization)所制定而发布的视频、音频、数据的压缩标准。

MPEG 标准的视频压缩编码技术主要利用具有运动补偿的帧间压缩编码技术以减小时间冗余度，利用 DCT 技术以减小图像的空间冗余度，利用熵编码则在信息表示方面以减小统计冗余度。这几种技术的综合运用，大大增强了压缩性能。总体来说，MPEG 在三方面优于其他压缩/解压缩方案，首先，由于在一开始它就是做为一个国际化的标准来研究制定，所以具有很好的兼容性。其次，MPEG 能够比其他算法提供更好的压缩比，最高可达 200:1。更重要的是，MPEG 在提供高压缩比的同时，对数据的损失很小。所以在多媒体数据压缩标准中，较多采用 MPEG 系列标准，包括 MPEG-1、2、4 等[3][4]。如表 1-1 所示：

表 1-1 MPEG 系列标准

标准简称	标准全称	批准时间
MPEG-1	最高约 1.5Mpbs 数字存储媒体的运动图像及伴音编码（coding of Moving Picture and associated Audio for Digital Media at up to 1.5Mbps）标准	1988 年开始制定，1992 年 11 月通过，作为 ISO/IEC11172 号文件
MPEG-2	运动图像及伴音编码（coding of Moving Picture and associated Audio）标准 视频码率（3Mbps～100Mbps）	1990 年开始制定，1994 年 11 月通过，作为 ISO/IEC13818 号文件或 ITU-TH.262 建议
MPEG-4	视音频对象的编码（coding of Audio-visual objects）标准 视频码率（5Kbps～5Mbps）	1993 年 7 月制定，1995 年 5 月通过，作为 ISO/IEC14496 号文件
	多媒体内容描述接口（Multimedia	1997 年 7 月制定，2001 年 12

MPEG-7	Content Description Interface）标准	月通过，作为 ISO/IEC15938 号文件
MPEG-21	多媒体框架（Multimedia Framework）标准	1991 年 10 月形成多媒体框架原理，2002 年 5 月制定

1.1.1 MPEG-1 标准

MPEG-1(ISO/IEC 11172)是 MPEG 组织于 1992 年提出的第一个具有广泛影响的多媒体国际标准[5]。其正式名称为“基于数字存储媒体运动图像和声音的压缩标准”，可见，MPEG-1 着眼于解决多媒体的存储问题。MPEG-1 用于传输 1．5Mbps 数据传输率的数字存储媒体运动图像及其伴音的编码，经过 MPEG-1 标准压缩后，视频数据压缩率为 1/100-1 / 200，音频压缩率为 1 / 6.5。它可提供每秒 30 帧 352*240 分辨率的图像，当使用合适的压缩技术时，具有接近家用视频制式（VHS）录像带的质量。MPEG-1 允许超过 70 分钟的高质量的视频和音频存储在一张 CD-ROM 盘上。VCD 采用的就是 MPEG-1 的标准，该标准是一个面向家庭电视质量级的视频、音频压缩标准。由于 MPEG-1 的成功制定，以 VCD 和 MP3 为代表的 MPEG-1 产品在世界范围内迅速普及。继成功制定 MPEG-1 之后，MPEG 组织于 1996 年推出解决多媒体传输问题的 MPEG-2 标准。

1.1.2 MPEG-2 标准

该标准于 1996 年推出，拟解决多媒体传输问题。其正式名称为“通用的图像和声音压缩标准”。包括编号为 13818-1 系统部分、编号为 13818-2 的视频部分、编号为 13818-3 的音频部分及编号为 13818-4 的符合性测试部分。设计初衷就是实现高级工业标准的图象质量以及更高的传输率。由于 MPEG-2 在设计时的巧妙处理，使得大多数 MPEG-2 解码器也能解码播放 MPEG-1 格式的数据。

同时，由于 MPEG-2 的出色性能表现，已能适用于 HDTV，使得原打算为 HDTV 设计的 MPEG-3，还没出世就被抛弃了。(MPEG-3 要求传输速率在 20Mbits/sev-40Mbits/sec 间，但这将使画面有轻度扭曲)。除了做为 DVD 的指定标准外，MPEG-2 还可用于为广播，有线电视网，电缆网络以及卫星直播(Direct Broadcast Satellite)提供的广播级数字视频。MPEG-2 的另一特点是，其可提供一个较广的范围改变压缩比，以适应不同画面质量，存储容量，以及带宽的要求。

MPEG-2 标准的成功之处是开发了通用的压缩编码系统，是一种以类（profile）和级（level）为基础的规范化体系，这样可满足不同的图像分辨率和处理速度的需要。因此，MPEG-2 标准能广泛应用于卫星广播业务（BSS）、电缆电视（CATV）、数字电视地面广播、数字声音广播、多媒体终端等众多领域。其技术规范包括 6 类（profile）和 4 级（level），并采用分级编码。

所谓类是指 MPEG-2 的不同处理方法，每一个类都包括压缩和处理方法的一个集合。不同的类意味着使用不同集合的码率压缩工具，越高的类编码越精细，而每升高一类将提供前一类没有使用的附加工具，当然实现的代价会更高。

而解码器却是向下兼容的，任何一种高级类解码器均能解码低级类方法编码的图像。

表 1-2 MPEG-2 的编码类定义

编码类型	支持算法
简单类（Simple Profile）	I 帧、P 帧，支持 4：2：0 格式，无可测量性
主类（Main Profile）	简单类类型增加支持 B 帧格式，无可测量性
信噪比可分级（SNR Scalable）	主类类型增加支持信噪比可测量性
空间可分级（Spatial Scalable）	信噪比可分级类型增加支持空间可测量性
高级类	空间可分级增加支持 4：2：2
主类扩展类	主类类型的高码率扩展

所谓级是指 MPEG-2 的输入格式，标识从有限度的 VHS 质量图像到 HDTV 图像，每一种输入格式编码后都有一个相应的范围：

(1) 低级 LL（Low Level）：图像输入格式的象素是 ITU-RRec.BT601 格式的 1/4，即 352×240×30 或 352×288×25，相应编码的最大输出码率为 4Mbps。

(2) 主级 ML（Main Level）：图像输入符合 ITU-RRec.BT601 格式，即 720×480×30 或 720×576×25。相应编码的最大输出码率为 15Mbps，高级类 20Mbps。

(3) 1440 高级 H14L（High1440 Level）：是 1440×1152×25 的高清晰度格式。相应编码的最大输出码率为 60Mbps，高级类 80Mbps。

(4) 高级 HL（High Level）：是 1920×1152×25 的高清晰度格式。相应编码的最大输出码率为 80Mbps，高级类 100Mbps。

级与类之间的组合构成 MPEG-2 视频标准在某种特定应用下的一种技术规范，可以满足不同层次的应用需求，且具有良好的向下兼容性。本文研究的视频硬件解码器满足 MP@ML 的技术规范的的 VLSI 设计。

1.1.3 MPEG-4 标准

MPEG-4 在 1995 年 7 月开始研究，1998 年 11 月被 ISO/IEC 批准为正式标准，编号是 ISO/IEC14496，它是超低码率运动图像和语言的压缩标准[7]，用于传输速率低于 64kbps 的实时图像传输，它不仅可覆盖低频带，也可以向高频带发展。所以 MPEG-4 更适于交互 AV 服务以及远程监控，MPEG-4 传输速率在 4800-6400bps 之间，分辨率为 176×144，可以利用很窄的带宽通过帧重建技术压缩和传输数据，从而能以最少的数据获得最佳的图像质量。因此，它将在数字电视、动态图像、互联网、实时多媒体监控、移动多媒体通信、Internet / intranet 上的视频流与可视游戏、DVD 上的交互多媒体应用等方面大显身手。

最重要的是，较之前两个标准而言，MPEG-4 为多媒体数据压缩提供了一个更为广阔的平台。它更多定义的是一种格式、一种架构，而不是具体的算法。它可以将各种各样的多媒体技术充分用进来，包括压缩本身的一些工具、算法，也包括图像合成、语音合成等技术。MPEG-4 从其提出之日起就引起了人们的广泛关注，虽然不是每个人都清楚它的具体目标，但却都对它寄予了很大的希望。

MPEG-4 的最大创新在于赋予用户针对应用建立系统的能力，而不是仅仅使用面向应用的固定标准。此外，MPEG-4 将集成尽可能多的数据类型，例如自然和合成的数据，以实现各种传输媒体都支持的内容交互的表达方法。借助于 MPEG-4，我们第一次有可能建立个性化的视听系统。

1.1.4 MPEG-7/21 标准

MPEG-7 于 1996 年 10 月开始研究，制定这个标准的主要目的，是为了解决多媒体内容的检索问题。确切来讲，MPEG－7 并不是一种压缩编码方法， 它并不针对某个具体的应用，而是通过这个标准，MPEG 希望对以各种形式存储的多媒体结构有一个合理的描述，通过这个描述，用户可以方便地根据内容访问多媒体信息。在 MPEG-7 体系下，用户可以更加自由地访问媒体。比如，用户可以在众多的新闻节目中寻找自己关心的新闻，可以跳过不想看的内容而直接按自己的意愿收看精彩的射门集锦；在互联网上，用户键入若干关键词就可以在网上找到自己需要的克林顿的演讲、贝多芬的交响乐等；甚至用户只需出示一张克林顿的照片或哼一首音乐的旋律，都可以找到自己所需要的多媒体材料。所有这些，都取决于 MPEG-7 中对各种多媒体内容的描述。

MPEG 在 1999 年 10 月的 MPEG 会议上提出了“多媒体框架”的概念，同年的 12 月的 MPEG 会议确定了 MPEG-21 的正式名称是“多媒体框架”或“数字视听框架”，它以将标准集成起来支持协调的技术以管理多媒体商务为目标，目的为多媒体传输和使用定义一个标准化的、可互操作的和高度自动化的开放框架。

1.2 MPEG-2 与 VLSI 设计

视频编解码器实现的结构可分为可编程结构和专用结构。可编程结构是指设计使用一个执行指令的硬件核（一般为 DSP 或 ARM）来实现解码的实现[8][9]，通过在硬件核上运行程序完成解码功能，适用范围广，能随不同的算法提供支持，易于升级，缺点是电路规模的庞大和功耗的增加。专用结构一般是通过 ASIC 的方法设计专用的处理芯片，其适用范围较窄，但因为其卓越的性能（如数据处理效率较高，面积小和功耗低等）被越来越多的视频处理器所采用。所以随着国际压缩标准的不断提出，许多 VLSI 厂商看到了视频编解码芯片在视听工业、多媒体通信、广播等领域应用的广泛前景，纷纷开始了专用视频压缩和解压缩芯片的开发工作，自 1990 年 7 月 C-CUBE 公司开发出第一块 JPEG 专用芯片、于 1991 年又推出将 MPEG-1 视频解码器 CL450 以来，这一领域的 VLSI 设计在短短的十年间取得了长足的进步。

随着 1994 年 MPEG-2 标准的制定， VLSI 设计又集中到了 MPEG-2 标准上面，成为这十几年设计的主流，但其设计思想和方法却有着日新月异的变化。其大致的发展过程[10]为，第一阶段：解码系统分别由视频解码、音频解码和系统控制 3 个单独的芯片组合完成，以松下电气的 MN67740 芯片为例，它仅是一块视频解码芯片，其内含音频/视频数据包、视频解码、视频接口、存储器接

口等电路，主要功能是将输入的压缩数据解压为音频和视频数据信号，并只对视频数据进行解码输出。这类芯片在早期的 DVD 机上应用极为广泛；第二阶段：视频和音频解码在一块芯片上完成，系统控制由另一块芯片完成，整个 MPEG-2 的系统解码由这两块芯片构成，这类芯片以 LSI Logic 公司的 L64020 为代表，它于 1998 年推出，集成了两个独立的音视频解码器：音频解码器支持 ISO 13818-3 两声道解码、杜比数字 5.1 声道和线性 PCM 音频数据流，视频解码支持 ISO 13818-2 标准，解码分辨率可达到 720*480/30fps，这类芯片主要也是用在 DVD 上。第三阶段就是由一块芯片完成 MPEG-2 解码的全部功能，除此之外，现在，其设计思路还向着功能多样化、更高集成度、软硬件合成设计的方向发展。比如富士通 08 年 12 月推出的多路解码器芯片 MB86H01 就代表了这种设计思想：

表 1-3 MB86H01 解码器的主要规格表

内部 CPU		ARC ARM7TDMI - S™ (202.5MHz)； 用于 H.264 视频解码
视频	解码等级	MPEG-2 视频：主要类@主要级（MP@ML）解码器 H.264 视频：主要类/level3.0 解码器
	解码支持	ITU-R BT.656 输入，数字 RGB888 输出，YCrCb 模拟 SD 输出 支持 PAL / NTSC / SECAM 格式
音频	格式	MPEG-1 / 2 Layer I / II
	通道	2 路通道
	接口	L/R 串行、I2S、S/P-DIF
TS 处理		MPEG-2 TS，3 个输入流，内置 DVB 解扰器，3DES 加密 / 解密
DDR 存储器接口		16bit DDR-SDRAM 135MHz，128Mbit～512 Mbit SDRAM
FLASH 存储器接口		支持串行闪存、NOR Flash、NAND Flash
USB		USB2.0 高速 OTG 控制器
外部输入 / 输出接口		UART、I2C、Smart Card*2、Rx / Tx、PWM、GPIO

由表可知，芯片的主要特性有：

1. 此芯片由两个 MPEG-2 解码器和一个 H.264 解码器构成，能够解码标清 MPEG-2 和 H.264 格式，不仅能够处理西欧的 MPEG-2 的标清广播，还能支持即将在东欧和俄罗斯实行的 H.264 格式标清广播，又能用于中国有限电视服务中的交互式视频点播（VOD）。此外，两个 MPEG-2 解码器可同时处理 2 个视频流，观众可同时观看两个节目。

2. 芯片内嵌了 202.5MHz 的高性能 ARC CPU，并支持必要的音频解码、接受信号功能和屏幕显示功能，方便客户开发各式各样的应用。

3. 小封装，实现了低功耗，是便携式设备和小尺寸的理想之选。该解码器还集成了高速 USB2.0 OTG 控制器，极大的提高了与数码相机等外部设备的连接性。

由于集成度不高，所以第一阶段所属的芯片几乎消失在主流的应用市场，由

于市场的细分，第二类的产品还占有较大的应用市场，在更高集成度的产品上，取而代之的是第三阶段的各类的芯片产品。随着技术的进步和新理论的出现，芯片设计的实现方法越来越多样化，相信以后会有更多高效的 MPEG-2 解码芯片出现。

1.3 MPEG-2 的未来

MPEG-2 标准颁布以来已有 15 年了，所以很多人称之为陈旧的 MPEG-2，相比较而言，人们更沉醉于 MPEG-4 或 H.264 带来的优点。尽管它们给我们带来了一系列的变化，但在 DVD 和 DTV 领域，MPEG-2 技术仍然有很强的生命力。

虽然，新兴的 AVC 将在一些非传统的领域，例如移动手持设备和个人媒体播放器开始兴起，显然 AVC（H.264）正在创造一个更大的基于标准的数字视频市场，但 15 年的大众市场积累使得基于 MPEG-2 的内容和基础设施巩固了 MPEG-2 的地位。此外，DVD 设备和数字电视接收器已经进入了家庭，所以尽管大量的视频光盘和电视接收器支持 AVC，但这些设备依然需要支持 MPEG-2 以保证能够和已有的 MPEG-2 内容兼容。所以说，MPEG-2 编解码器在未来 10 年内会依然占据统治地位。

1.4 主要工作及论文结构

本论文完成的主要工作有以下方面：

1．建立了 MPEG-2 视频解码的软件模型，并给予解析改写，和硬件设计的结果作比较，以验证硬件解码结果的正确性。

2．实现了 MPEG-2 视频器的系统架构，并完成了大部模块的 RTL 级设计，并予以仿真。

3．搭建了解码器 RTL 级的验证平台，实现系统的 FPGA 原型验证，最后完成系统的逻辑综合，给出综合结果。

论文的结构安排如下：

第一章：绪论，介绍 MPEG 标准发展的历程和 MPEG-2 的 VLSI 设计史，并概括整体论文结构。

第二章：MPEG-2 的算法标准。本章对 MPEG-2 视频的编解码原理、解码的关键算法做了详细的阐述。

第三章：MPEG-2 视频解码的软件模型和硬件设计概述。本章利用 C 程序软件为要设计的硬件解码器建立了完整的参考，以便更早的发现设计中的问题。

第四章：MPEG-2 视频解码器的硬件实现。章实现 MPEG-2 视频解码器的 RTL 级代码设计，并通过仿真。

第五章：解码器的验证和综合。对完成设计的解码器进行 FPGA 验证，然后基于 smic0.18 工艺库完成逻辑综合。

最后总结全文，并展望了进一步的工作方向。

1.5 本章小结

本文首先介绍了国际上比较通用的 MPEG 压缩标准的发展过程，并对 MPEG-2 标准做了相对详细的介绍，确定实现 MPEG-2(MP@ML)视频解码器的 VLSI 设计。接着介绍了用 VLSI 实现 MPEG-2 解码的发展过程和 MPEG-2 的未来，最后概括了论文的结构，是整篇论文的起点。

第二章 MPEG-2 的算法标准

2.1 MPEG-2 解码原理

视频解码是从编码压缩码流中恢复视频对象的过程[12]。图 2-1 所示的是 MPEG-2 的编解码算法模型。输入视频序列首先做预处理，判断是否需要运动估计，它是指当前图像中象素块从参考帧中寻找最佳匹配块的过程，搜寻结果采用运动向量表示。根据运动向量从参考帧中形成对应的预测块，得到一个两维的矩阵预测采样值，将当前的图像减去预测值，剩余残值做离散余弦变换，并对系数进行量化操作，量化后的数据做可变长编码，得到最终的压缩数据。如果不需要运动估计，图像将做帧内编码类型的处理，依次做离散余弦变换、量化和可变长编码，最后形成码流。

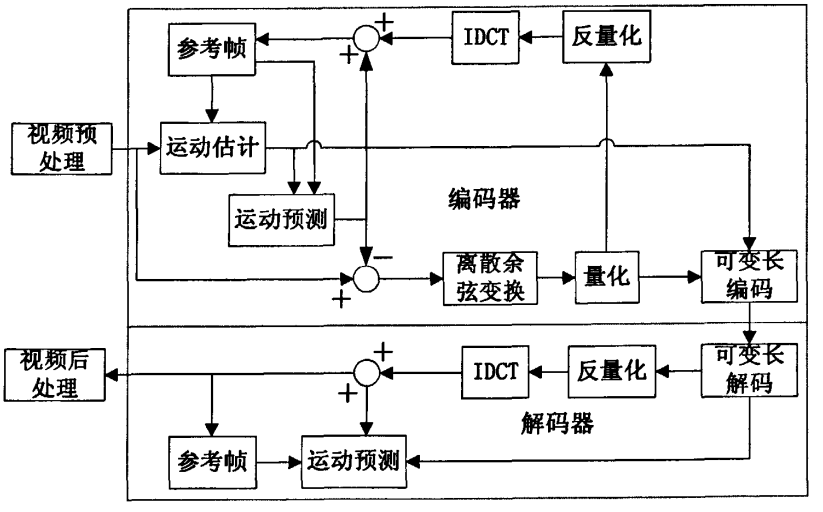

图 2-1 MPEG-2 编解码算法框图

MPEG-2 视频解码器是根据 MPEG-2 的视频解码算法对编码码流进行解码的系统。实现算法的解码有多种实现形式，比如有软件实现的解码（在下一章做详细介绍），有使用 ARM、DSP 等处理器编程实现解码的方法，也有采用专用视频解码器的方法，即设计出专用的硬件电路对码流进行解码。

在本文研究的设计中，就是实现对专用视频解码器的设计，其优点是可以完全按照特定的算法进行解码，最大程度的和算法相匹配，从而获得最高的处理速度，且功耗较低。本文的设计目标实现基于 MP@ML 层次的 MPEG-2 视频解码器。

2.1.1 MPEG-2 视频源数据的结构

编码的视频数据是由称作“层”的比特流序列组成，通常这些视频比特流可看作语法的等级体系，其中一个语法结构包含一个或更多的从属结构。例如，结构“picture_data()”包含一个或更多的语法结构“slice()”，而“slice()”又包

含一个或更多有序的结构“macroblock()”。

编码的比特流中的最高语法结构就是视频序列。一个视频序列以一个序列头开始，后面可选地跟着一组图像的头和一个或更多的编码帧。帧在编码比特流中的顺序就是解码器处理它们的顺序，但并不一定就是显示顺序。视频序列以一个 sequence_end_code 终止。在一个视频序列的不同地方，某一特定的编码帧的前面可能会有一个重复的序列头或一组图像的头，或两者都有，具体的语法结构如下表所示。

表 2-1 视频序列结构

video_sequence(){	位数
next_start_code()	
sequence_header()	
if (nextbits()==extension_start_code){	
sequence_extension()	
do{	
extension_and_user_data(0)	
do {	
if (nextbits()==group_start_code){	
group_of_pictures_header()	
extension_and_user_data(1)	
}	
picture_header()	
picture_coding_extension()	
extensions_and_user_data(2)	
picture_data()	
}while((nextbits()==picture_start_code) ||	
(nextbits()==group_start_code))	
if (nextbits()!=sequence_end_code){	
sequence_header()	
sequence_extension()	
}	
} while(nextbits()!=sequence_end_code)	
} else{	
/* ISO/IEC 11172-2 */	
}	
sequence_end_code	32
}	

MPEG-2 的编码码流分为六个层次，如图 2-2 所示，从上至下依次为：视频序列层(sequence)，图组层(GOP：group of picture)，图像层（picture），组块层（slice），宏块层（Macroblock）和块层（block）。除宏块层和块层，上面四层中都有相应的起始码，它们可用于因误码或其他原因收发两端失步时，解码器可以重新捕捉同步。

视频序列指构成某路视频节目的图像序列，一个视频码流可以由一个或多个

序列组成。序列起始码后的序列头中包含了图像尺寸、象素宽高比、帧频、码率、帧组数等信息，序列扩展中包含了一些附加数据。为保证能随时定位图像序列，序列头是重复发送的。

序列层下是图组层，这一层的头信息用在一个编码 I 帧（后面介绍）的前面，来向解码器指明在随机访问时，紧跟在编码 I 帧后面的第一个 B 帧是否能被正确重构。实际上，如果不能得到前面的参考帧，那些 B 图就不能被正确重构，除非它们仅使用后向预测。一个图组由相互间有预测和生成关系的一组 I、P、B 图像构成，但头一帧图像总是 I 帧。图组头中包含了时间信息。

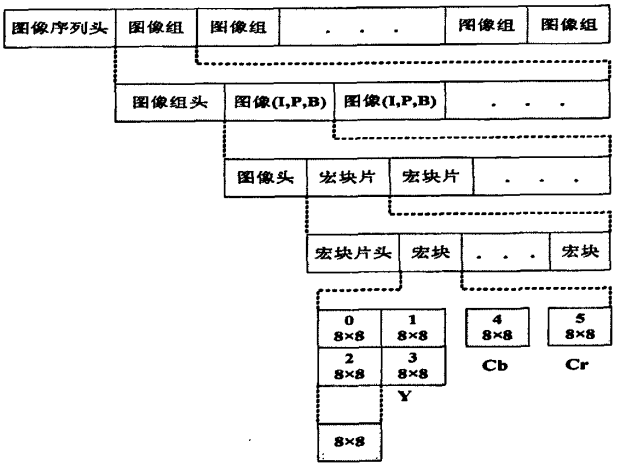

图 2-2　MPEG-2 视频流分层结构

图组层下是图像层，此层是图像编码的基本单元。分为 I、P、B 三类，包含了帧类型、帧编号等信息。

图像层下是组块层，此层用作再同步单元。一个组块是一系列任意数目的宏块，至少要包含一个宏块。组块的第一个和最后一个宏块必须在同一宏块水平行。组块间不能重叠。组块按它们将要被遇到的顺序出现在比特流中，从图像的左上角开始，按从左到右，从上到下光栅扫描顺序排列。

组块层下是宏块层。宏块包含一部分亮度分量和空间相关的色差分量。MPEG-2 中定义了三种宏块结构：4：2：0 宏块、4：2：2 宏块和 4：4：4 宏块，分别代表构成一个宏块的亮度像块和色差像块的数量关系：4：2：0 宏块中包含四个亮度像块，一个 Cb 色差像块和一个 Cr 色差像块；4：2：2 宏块中包含四个亮度像块，二个 Cb 色差像块和二个 Cr 色差像块；4：4：4 宏块中包含四个亮度像块，四个 Cb 色差像块和四个 Cr 色差像块。这三种宏块结构实际上对应了三种亮度和色度的抽样方式。

宏块层下是块层，它是 MPEG-2 视频码流的最低层，是 DCT 变换的基本单元。MP@ML 中一个块是基于 8×8 个抽样值构成，同一个块内的抽样值必须全部都是 Y/Cb/Cr 信号样值。另外，块也应用于表示 8×8 个抽样值经 DCT 变换后所生成的示 8×8 个 DCT 系数。

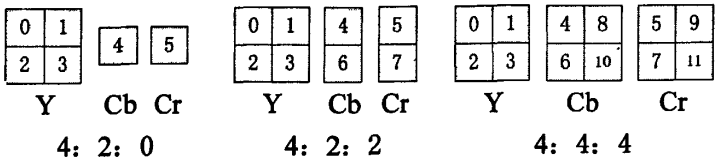

图 2-3 三种不同宏块结构图

2.1.2 帧类型

MPEG-2 视频压缩的原理是利用了图像中的两种特性：空间相关性和时间相关性，这两种相关性使得图像中存在大量的冗余信息。为了能够去掉图像中的冗余信息，MPEG-2 定义了三种编码帧类型，分别是 I 帧、P 帧和 B 帧。

三种帧采用不同的编码原则：I 帧采用帧内编码方式，其帧中所有宏块的编码方式都是帧内编码(intra 模式，表示数据的编码跟其他帧图像的数据无关)。P 帧和 B 帧采用帧间预测方式，P 帧图像采用前向时间预测，一部分宏块是帧间编码(帧间模式，表示宏块内的编码数据和参考帧的数据有关)，另一部分是帧内编码，具体的编码方式由运动搜索的结果决定。B 帧图像采用双向时间预测，宏块的编码方式只有帧间模式。

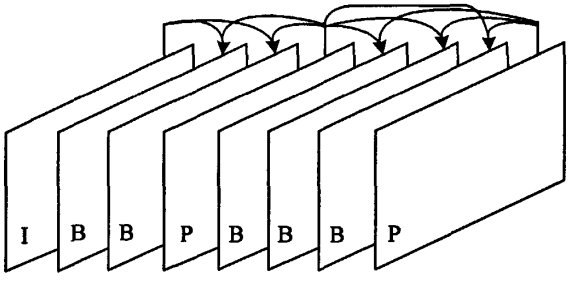

图 2-4 MPEG-2 帧类型和预测方式

下面来介绍三种帧图像的主要特点。

I 帧：

- 一个完整的帧内压缩编码帧，解码时仅用 I 帧的数据就可重构完整图像。
- I 帧描述了图像背景和运动主题的详情。
- I 帧不需要参考其他画面生成，不需要考虑运动矢量。
- 是 P 帧和 B 帧的参考帧，其质量直接影响到同组中以后各帧的解码质量。
- 是图组 GOP 的基础帧，每个图组的第一帧必须是 I 帧。
- 所占据的数据信息量在三种帧类型中是最大的。

P 帧：

- P 帧属于前向预测的帧间编码，只参考前面靠近它的 I 帧或 P 帧。
- 采用运动补偿的方法传送它与前面的 I/P 帧的差值及运动向量，解码时必须将参考帧的预测值与预测误差求和后才能重构完整的 P 帧图像。
- P 帧可以是后面 P 帧的参考帧，也可以是其前后 B 帧的参考帧。
- 由于 P 帧可能用作参考帧，所以它可能造成解码错误的扩散。

B 帧：

• 双向预测编码帧，可有前面的 I/P 帧和后面的 P 帧来进行预测。

• B 帧传送的是它与前面的 I 或 P 帧和后面的 P 帧之间的预测误差及运动矢量。

• B 帧压缩比最高，因为它只反映两参考帧间运动主体的变化情况，所以预测比较准确。

• 由于不是参考帧，不会造成解码错误的扩散。

2.1.3 帧排序规则

上面已经提到，B 帧是双向预测。在这种前提下，当编码帧队列中有 B 帧时，就要对视频序列进行重新排序，排序过程遵循以下规则：

• 如果编码序列中的当前帧是第一个 I 帧，则立即编码。

• 如果编码序列中的当前帧是 B 帧，则暂存该帧，先对该 B 帧的前后参考帧编码，这里的参考帧是指 I 帧或 P 帧。

• 编码序列中的 P 帧遵循第二条规则排序。

下面是一个带 B 帧的编码序列重新排序的例子，在这个例子中两个 P 帧或者一个 I 帧与一个 P 帧中存在两个 B 帧。按照排序规则，排序过程如下：

在编码器输入：

1	2	3	4	5	6	7	8	9	10	11	12	13
I	B	B	P	B	B	P	B	B	I	B	B	P

在解码器输入：

1	4	2	3	7	5	6	10	8	9	13	11	12
I	P	B	B	P	B	B	I	B	B	P	B	B

在解码器输出：

1	2	3	4	5	6	7	8	9	10	11	12	13

连续的编码 B 帧的数目是可变的，在顺序的编码 P 帧（或编码 I 帧和 P 帧之间）可能不出现 B 帧。每组的 B 帧在比特流中的顺序就是它们在解码输出端显示的顺序。如果序列中不包含编码 I 帧，则在序列的开始处、在序列中需要随即访问和错误复原的地方，都应特别注意。

2.2 MPEG-2 的关键技术

概括的说，MPEG-2 图像压缩的原理是利用了图像中的两种特性：空间相关性和时间相关性。一帧图像内的任何一个场景都是由若干象素点构成的，因此一个象素通常与它周围的某些象素在亮度和色度上存在一定的关系，这种关系叫做空间相关性；一个节目中的一个情节常常由若干帧连续图像组成的图像序列构成，一个图像序列中前后帧图像间也存在一定的关系，这种关系叫做时间相关性。这两种相关性使得图像中存在大量的冗余信息，如果我们能够将这些冗余信息去除，只保留少量非相关性信息进行传输，就可以大大节省传输频带。而解码器利用这些非相关信息，按照一定的解码算法，就可以在保证一定的图像质量的前提下恢复原始图像。一个好的压缩编码方案就是能够最大限度地去

除图像中地冗余信息，而一个好的解码算法就是尽可能的还原编码前的原始图象，MPEG-2 的一些关键技术如下：

2.2.1 离散余弦变换（DCT）

因为静态图像和预测误差信号两者具有非常高的空间冗余度，压缩系统的第一步工作就是识别存在于视频信号的每帧每场中的空间冗余，为降低空间冗余，最广泛是采用频域分解技术即 DCT。它将图像信息块转换成代表不同频率分量的系数集，这是一个无损、可逆的数学过程。

由于视频图像的自然属性，DCT 变换经常使得代表高空间频率的 DCT 系数数值很小。类似的，由于人类视觉分辨特点，许多较高空间频率的数值可以很粗糙地定义（就是用较少地 bit 来表示）或完全不用，也不会明显地影响解码图像的质量。所以经 DCT 变化后，大部分信号能量集中在少数几个系数上，这就使得只编码少数系数而不严重影响图像质量成为可能，也为下面的量化运算打下了基础。

由于大多数图像的高频分量较小，相应于图像高频分量的系数经常为零，加上人眼对高频成分的失真不太敏感，所以可用更粗的量化。因此，在视频信号数字处理时，可根据频谱因素分配比特数：对包含信息量大的低频谱区域分配较多的比特数，对包含信息量低的高频谱区域分配较少的比特数，而图像质量并没有造成可察觉的损伤，达到码率压缩的目的。虽然到达解码器后通过反离散余弦变换回到样值，虽然会有一定的失真，但人眼是可以接受的。

DCT 的进行是以像素块为基本单位的，可以是基于宏块的，也可以是基于亮度块或色度块的。在 MPEG-2 中 DCT 以 8x8 的象素块为单位进行，生成的是 8x8 的 DCT 系数数据块。公式 2-1 是 IDCT 的计算公式：

$$f(x,y)=\frac{2}{N}\sum_{0}^{7}C(u)C(V)F(u,v)\cos\frac{(2x+1)u\pi}{2N}\cos\frac{(2y+1)v\pi}{2N} \qquad (2\text{-}1)$$

其中，

$C(w)=\frac{1}{\sqrt{2}}$，当 w=0　　$C(w)=1$，当 w=1，2，．．7

对所有的 x，y 值，$f(x,y)$的值限于$-256\le f[y][x]\le 255$。DCT 虽然不能直接对图像产生压缩作用，但对图像的能量具有很好的集中效果，为压缩打下了很好的基础。

2.2.2 量化器

原图像经过 DCT 变换所得到的频率系数呈现出这样的特点：大部分系数为零或接近与于零，低频系数较大，高频系数较小。为了提高编码效率，需要对变换后的系数进行量化，一方面可以增加零系数的个数，另一方面可以减小非零系数的表示范围。量化过程就是以某个量化步长去除 DCT 系数。量化步长的

大小称为量化精度，量化步长越小，量化精度就越细，包含的信息越多，但所需的传输频带越高。不同的 DCT 变换系数对人类视觉感应的重要性是不同的，因此编码器根据视觉感应准则，对一个 8x8 的 DCT 变换块中的 64 个 DCT 变换系数采用不同的量化精度，以保证既尽可能多地包含特定的 DCT 空间频率信息，又使量化精度不超过需要。DCT 变换系数中，低频系数对视觉感应的重要性较高，因此分配的量化精度较细；高频系数对视觉感应的重要性较低，分配的量化精度较粗。

量化是压缩算法中产生失真的根源，量化器设计的好坏直接影响到重建图像的性能，解码过程中反量化的运算如图 2-5 所示。

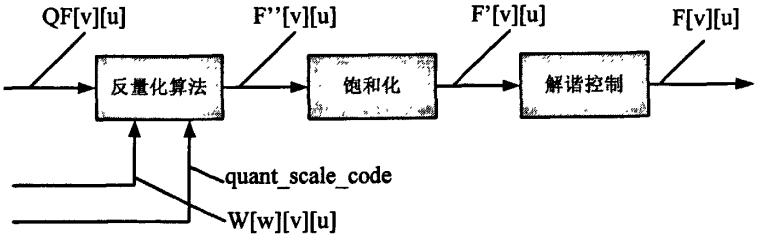

图 2-5 反量化过程

图 2-5 中 QF[v][u]为量化的 DCT 系数，F[v][u]为重构的 DCT 系数，quant_scale_code 为量化标尺，W[w][v][u]为量化权矩阵。系数的两维数组 QF[v][u]被反量化而重构 DCT 系数。这一过程的实质是以量化器步长为倍数的乘法运算，在适当的反量化算法之后，结果系数 F''[v][u]被饱和化而生成 F'[v][u]，之后执行一个解谐控制操作来给出最终的重构 DCT 系数 F[v][u]。

对于反量化所有系数来说，分为两种情况：

1．内部 DC（直流）系数，内部编码块的 DC 系数以一种不同于其他系数的方法进行反量化。方法之一是与一个常数（由 intra_dc_precision 直流精度系数决定）相乘，如公式 2-2 所示：

$$F''[0][0]=intra\_dc\_mult * QF[0][0] \qquad (2\text{-}2)$$

2．其他系数：与量化器比例因子和加权矩阵相乘，量化器比例因子由 quant_scale_type（决定是线性量化还是非线性量化）和 quant_scale_code 共同决定。用到了两种加权矩阵：用户自定义矩阵和缺省矩阵。如公式 2-3 所示：

$$F''[v][u]= ((2\,QF[v][u]+k)*W*quant\_scale)/32 \qquad (2\text{-}3)$$

$$K=\begin{cases} 0 & \text{内部块} \\ Sign(QF[v][u] & \text{非内部块} \end{cases}$$

从反量化算法中得到的结果系数被饱和化在[-2048:2047]的范围内，这就是饱和化算法，如公式 2－4 所示：

$$F''[v][u]=\begin{cases}2047 & F''[v][u]\geqslant 2047\\ F''[u][v] & -2048\leqslant F''[v][u]\leqslant 2047\\ -2048 & F''[v][u]<-2048\end{cases} \qquad (2\text{-}4)$$

解谐控制由以下等价的过程实现。首先，块中所有的这些重构的、饱和化的F'[v][u]被求和，然后检测这个值为奇还是偶。如果和为偶，则要对系数 F[7][7]进行调整修改。如公式 2-5 所示：

$$sum=\sum_{0}^{7}\sum_{0}^{7}F'[v][u]$$

$$F[7][7]=\begin{cases}F[v][u]=F'[v][u] & \\ F'[7][7] & sum\text{ 为奇}\\ \left.\begin{cases}F'[7][7]-1 & \text{如果 } F'[7][7]\text{为奇}\\ F'[7][7]+1 & \text{如果 } F'[7][7]\text{为偶}\end{cases}\right\} & sum\text{ 为偶}\end{cases} \qquad (2\text{-}5)$$

2.2.3 之型扫描与游程编码

DCT 变换产生的是一个 8x8 的二维数组，为进行传输，还须将其转换为一维排列方式。有两种二维到一维的转换方式，或称扫描方式：之型扫描(Zig-Zag)和交替扫描，其中之型扫描是最常用的一种。由于经量化后，大多数非零 DCT 系数集中于 8x8 二维矩阵的左上角，即低频分量区，之型扫描后，这些非零 DCT 系数就集中于一维排列数组的前部，后面跟着长串的量化为零的 DCT 系数，这些就为游程编码创造了条件。Zig-zag 扫描的定义和交替扫描的定义分别如图 2-6 所示：

$$\begin{pmatrix}0&1&5&6&14&15&27&28\\2&4&7&13&16&26&29&42\\3&8&12&17&25&30&41&43\\9&11&18&24&31&40&44&53\\10&19&23&32&39&45&52&54\\20&22&33&38&46&51&55&60\\21&34&37&47&50&56&59&61\\35&36&48&49&57&58&62&63\end{pmatrix}\qquad\begin{pmatrix}0&4&6&20&22&36&38&52\\1&5&7&21&23&37&39&53\\2&8&19&24&34&40&50&54\\3&9&18&25&35&41&51&55\\10&17&26&30&42&46&56&60\\11&16&27&31&43&47&57&61\\12&15&28&32&44&48&58&62\\13&14&29&33&45&49&59&63\end{pmatrix}$$

图 2-6 反扫描的方式

在 MPEG-2 标准的变长码部分除了使用 huffman 编码之外，还使用了游程长度编码（Run Length Encoding）。游程编码中，只有非零系数被编码。一个非零系数的编码由两部分组成：前一部分表示非零系数前的连续零系数的数量(称为游程)，后一部分是非零系数。这样就把之型扫描的优点体现出来了，因为之型扫描在大多数情况下出现连续零的机会比较多，游程编码的效率就比较高。当一维序列中的后部剩余的 DCT 系数都为零时，只要用一个“块结束”标志(EOB)来指示，就可结束这一 8x8 变换块的编码，产生的压缩效果是非常明显的。

2.2.4 熵编码

量化仅生成了 DCT 系数的一种有效的离散表示，实际传输前，还须对其进行比特流编码，产生用于传输的数字比特流。简单的编码方法是采用定长码，即每个量化值以同样数目的比特表示，但这种方法的效率较低。而采用熵编码可以提高编码效率。熵编码是基于编码信号的统计特性，使得平均比特率下降。游程和非零系数既可独立的，也可联合的作熵编码。

熵编码中使用较多的一种是霍夫曼编码，MPEG-2 视频压缩系统中采用的就是霍夫曼编码。霍夫曼编码中，在确定了所有编码信号的概率后生产一个码表，对经常发生的大概率信号分配较少的比特表示，对不常发生的小概率信号分配较多的比特表示，使得整个码流的平均长度趋于最短。具体实现霍夫曼编码的基本步骤如下：

⑴ 将信源符号出项的概率由大到小排列。

⑵ 将两处最小的概率进行组合相加，形成一个新概率：并按第一步的方法重排，将如此重复进行直到只有两个概率为止。

⑶ 分配码字，码字分配从最后一步开始反向进行，对最后两个概念一个赋予‘0’码字，一个赋予‘1’码字。如此反向进行到开始的概率排列，在此过程中，如果概率不变采用原码字。

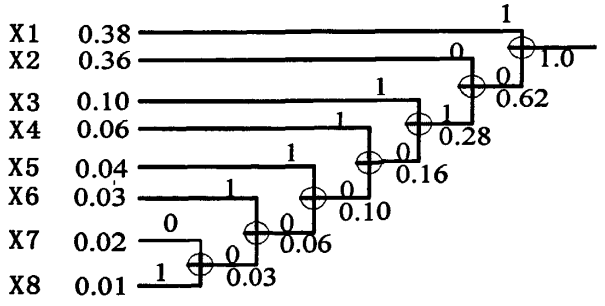

图 2-7 Huffman 编码

按照上述步骤，可以得出各字符的编码码字，X1：1；X2：00；X3：011；X4：0101；X5：01001；X6：01000；X7：0100010；X8：0100011。由此可以计算出平均码长为：

(1*0.38)+(2*0.36)+(3*0.10)+(4*0.06)+(5*0.04)+(6*0.03)+(7*0.02)+(8*0.01)=2.23。码长缩小 3-2.23＝0.77，节省了 26%的存储空间。

2.2.5 运动估计

运动估计使用于帧间编码方式时，通过参考帧图像产生对被压缩图像的估计，即求出从上一帧到当前帧的方向和象素的运动向量。运动估计的准确程度对帧间编码的压缩效果非常重要，如果估计作的好，那么被压缩图像与估计图

像相减后只留下很小的值用于传输。运动估计以宏块为单位进行，计算被压缩图像与参考图像的对应位置上的宏块间的位置偏移。这种位置偏移是以运动矢量来描述的，一个运动矢量代表水平和垂直两个方向上的位移。运动估计时，P 帧和 B 帧图像所使用的参考帧图像是不同的。P 帧图像使用前面最近解码的 I 帧或 P 帧作参考图像，称为前向预测；而 B 帧图像使用两帧图像作为预测参考，称为双向预测，其中一个参考帧在显示顺序上先于编码帧(前向预测)，另一帧在显示顺序上晚于编码帧(后向预测)，B 帧的参考帧在任何情况下都是 I 帧或 P 帧。

运动估计算法可以归为两类：第一类称为块匹配算法，这个算法假定一个图像块中的所有象素做同一运动，所以运动估计可以基于一定的误差从前一帧中寻找到最相似的子块，从而得到最佳位移向量。另一类算法是递归算法，如果连续帧中，象素数据的变化是因为物体的位移引起的，算法就会在梯度方向做叠代运算，使连续的运算最后收敛于某个运动估计向量。虽然块匹配算法存在缺点，即图像被分解成互相独立编码的数据块，做 DCT 变换时产生方块效应。但因块匹配算法简单，位移跟踪能力强，且易于大规模集成实现，所以得到了广泛应用。

在块匹配算法中，每帧图像被分成二维的 N*N 象素的子块,在 MPEG-2 标准中，一般以宏块作为基本的的子块（为 16×16 象素），假设每个子块内的象素都作相同的平移运动，当前帧的 N*N 子块在参考帧对应的子块相邻区域内搜索与之最匹配的块，当前子块与匹配块在二维平面上的位移即为运动故估计得到的运动向量，如图 2-8 所示。

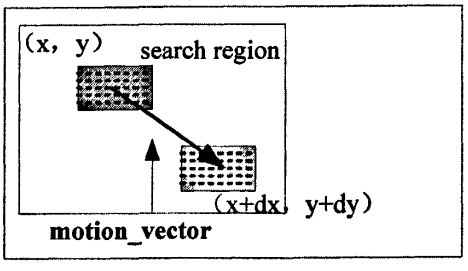

图 2-8 运动估计与运动向量的几何关系

运动估计算法的研究从两方面着手：快速搜索算法和块匹配准则。

块匹配的准则决定何时找到最佳匹配块，从而终止搜索进程。传统的准则主要有均方误差准则(MSE)、绝对平均误差准则(MAE)、互相关函数(CCF)、最大误差最小函数(MME)等。其中，MME 匹配函数过于简单，没有充分利用匹配块所包含的特征信息，使得运动估计的精度大大降低；CCF 匹配函数的计算过于复杂；与 MSE 相比较，MAE 匹配函数的计算量相对较小，而效果却与 MSE 相近，因而得到广泛应用。绝对平均误差准则(MAE)的表达公式为：

$$MAE(i,j)=\frac{1}{N^2}\sum_{0}^{N-1}\sum_{0}^{N-1}|C(x+i,y+j)-R(x+i+1,y+j+1)| \qquad (2\text{-}6)$$

有了匹配准则，剩下的问题就是寻找最优匹配块的搜索方法。最简单的搜索法是全搜索法，它穷尽参考帧搜索窗内所有可能的点进行比较，因此，一般来说，它的精度最高，但其计算量较其他方法过于庞大，限制了其应用的范围。而三步搜索由于简单、健壮和性能良好的特点，为设计人员所重视。此算法的基本思想是采用一种由粗到细的搜索方式，从原点开始，按一定的步长取周围 8 个点构成每次搜索的点群，然后进行匹配计算，跟踪最小误差的点。然后缩短步长，重复上述操作，连续三次即可完成。

此外，还有其他的快速算法，如共轭方向搜索法、交叉搜索法、新三步搜索法等，这里就不详细介绍了。

2.2.6 运动补偿

运动估计和运动补偿是紧密联系的，可以说，运动补偿实际上就是运动估计的逆过程，运动补偿假定：当前帧中的图像块是参考帧的图像块的某种平移，这就为使用预测和内插提供了机会。其基本原理可简述如下：当编码器对图像序列中的第 N 帧进行运动估计，得到第 N 帧的预测帧 N'。在实际编码传输时，并不总是传输第 N 帧，而是第 N 帧和其预测帧 N' 的差值△N，如果运动估计十分有效，△N 值的分布概率基本上在零的附近，这样需传输的比特数就大为减少，这就是运动补偿技术能够去除信源中的时间冗余的本质所在。概括来说，它就是利用运动估计算出的运动矢量，将参考帧图像中的宏块移至水平和垂直方向上的对应位置，即可生成对被压缩图像的预测。在绝大多数的自然场景中运动都是有序的，因此这种运动补偿生成的预测图像与被压缩图像的差分值是很小的。

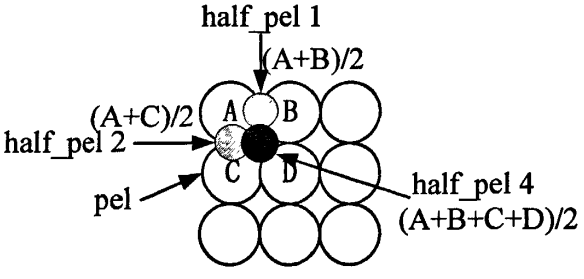

图 2-9 半象素插值

MPEG-2 标准采用半象素运动补偿策略，显著提高了图像质量。半象素值通过双线性内插法(取周围最近的整象素值)得到，如图 2-9 所示，在运动矢量搜索之前需要对宏块数据进行插值操作。由于色度分辨率要求不高，在运动搜索时，只对亮度进行操作。按照取周围最近的半象素位置的原则，不同色度块的运动矢量由运动矢量的值除以二得到。

对于 MPEG-2 中的 B 帧，还有双向插值模式，即 B 帧的前向参考帧和后向参考帧之间进行插值。具体插值方法就是将前向参考帧和后向参考帧的插值结果 A、half-pel1、half-pel2、half-pel3 再进行求平均运算。

2.3 本章小节

本章先对视频编解码器所经常用到的技术做了一些原理方面的介绍，接下来介绍视频编解码器系统的一般组成和现有的视频标准，最后对 MPEG-2 视频解码器中的一些关键技术进行了讨论。在本章的基础上，在下一章对解码器的软件模型进行分析并提取其中的算法。

第三章 MPEG-2 视频软件算法和硬件设计概述

3.1 概述

在 MPEG-2 视频解码的硬件设计中，对模块进行 RTL 级语言描述时，若要保证 MPEG-2 视频解码每一步骤如：huffman、IQ、IDCT 等的正确性，需要用到参考软件验证来实现。它实际上就是用纯软件实现用硬件实现的目标，而又由于软件使用的普遍性和广泛性，它的获得比较容易。所以对硬件设计来说，它是一个很好的参考。

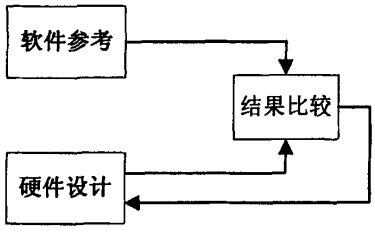

图 3-1 参考软件作用

如图中所示，当我们硬件设计完成时，要检测其模块功能是否正确，所以要和参考软件的结果进行比对，若结果一致，则代表硬件设计正确。比如，IDCT 模块设计完成后，拿模块的 64 个输出数据和参考软件的 IDCT 运算结果进行比较，若两者结果一致，则表示硬件模块设计的功能正确，若不一致，则需返回硬件模块，找出错误的原因，修改硬件的设计，直至结果完全相同。

MPEG-2 标准的发布已经有相当长的时间，对 MPEG-2 标准的研究也取得了很好的进展。在众多研究的组织及团体中，各有自己的解码算法。在本次设计中，采用的是 www.MPEG.org 的开放 MPEG-2 解码算法作为参考模型的算法，相对于其他组织的视频解码代码而言，此模型有较为完整的视频解码算法，且算法在结构上较为清晰，对于各个函数的功能实现也有较为全面的注解，所以说，此参考软件的原始代码具有很强的可读性，这对我们的理解和应用都有很大的方便。此外，此代码在功能上也较为完善，对于各种不同采样结构的编码方案均有简洁有效的解码过程。

整个参考软件的代码中包含各个层次分级的解码方法，由于各个分级的解码方式相同，且基于目前最通用的 4：2：0 的样本结构，本次设计采用本结构的数据流进行解码分析。

3.2 参考软件的控制实现

软件中的算法依功能划分，解码系统由 9 大功能性模块组成，依数据流的流向排列为：取数据及数据调整（getbits）；解序列数据（gethdr）；解图象组（getpic）；解变长码参数（getvlc）；解块变长码（getblk）；反离散余弦变换（IDCT）；解运动向量（motion）；图像预测补偿（recon），显示（display）。

另外，还需要其他模块来协调功能性模块的工作，即功能控制（mpeg2dec）、空域分级控制（spatscal）、存储（store）、帧数据缓冲控制（subspic）和系统控制（systems），

因为编码码流数据的结构是按层组成，所以其解码过程就可以分成是各个不同层数据的解码，这就形成了解码系统的解码方式，MPEG-2 视频解码码系统的控制流程具体实现过程图 3-2 所示。

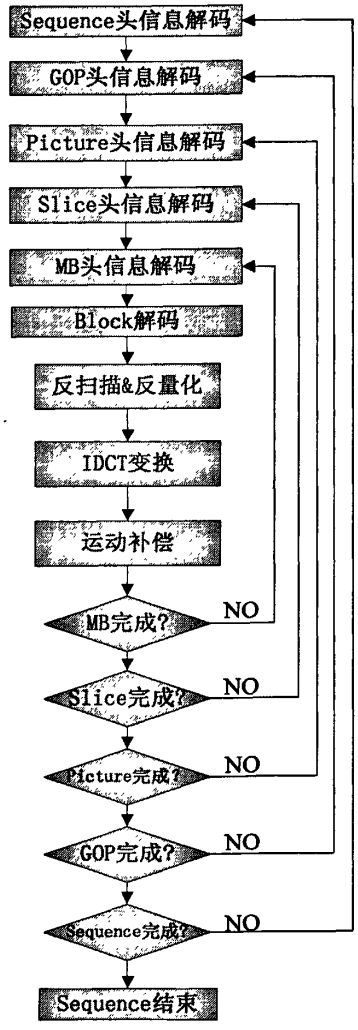

图 3-2 软件解码流程

要实现图中描述的解码过程，软件程序必须包含实现这些功能的函数，本章内容只介绍实现图像解码的主体函数之间的关系，其余的就不详细介绍了[13]。

顶层解码函数之间的关系如图 3-3 所示，其中 Option_initialize()，Initialize_Buffer ()，Initialize_Decoder()是对整个软件的初始化工作。

Decode_Bitetream()函数为整个解码系统的主体解码函数。Headers()为解序列头函数；video_sequence()为主体解图像函数，这两个函数的循环执行就可解

码一个图像。video_sequence()中除了初始化之外，函数 Decode_Picture()为图像解码的主体部分。

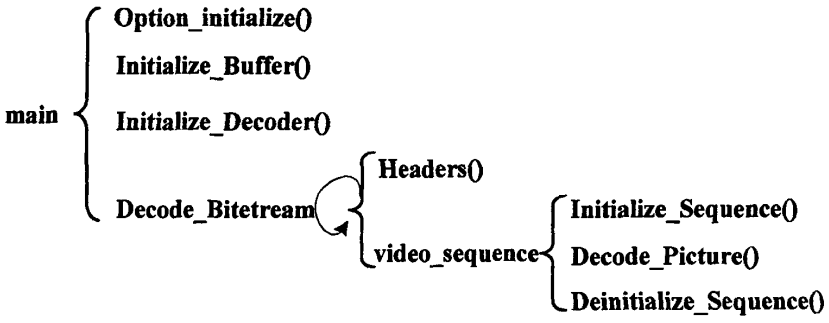

图 3-3 主程序软件结构图

在上图中的 headers()函数主要实现到 slice 层之前的序列数据解码，主要在 gethdr.c 程序中完成。图中的 Decode_Picture()函数在程序 getpic.c 中实现，其解码的具体结构如图 3-4 所示。

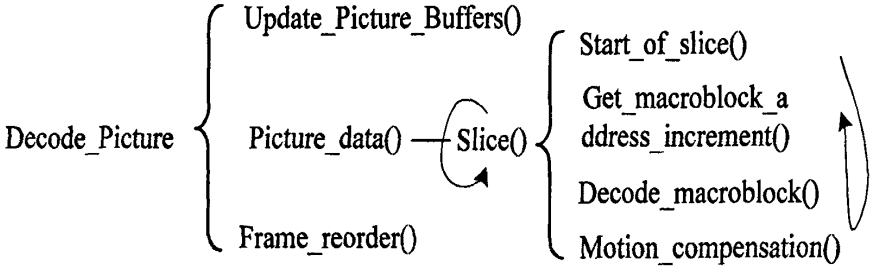

图 3-4 getpic 的解码结构

其中，Update_Picture_Buffers()函数和 frame_reorder（）函数的功能主要是基于 I、P、B 图像的重排工作，在第二章，我们已知道数据流中的图像数据和解码后的显示数据不同，需要进行重排，此外，frame_reorder 还进行解码数据存储和显示工作。而 Picture_data()是 getpic 程序中的主体解码函数，一个 picture 由若干个 slice 组成，picture 是否解码结束，由解完的 slice 的结果决定。一个 slice 中又含有若干个宏块（MB），所以 slice 函数中又包含如下函数：start_of_slice()函数进行 slice 的头参数解码，Get_macroblock_address_increment()函数决定是否存在跳跃的宏块，decode_macroblock()函数执行对宏块的解码，motion_compensation()进行运动预测补偿。这三个函数每次执行一个宏块的解码，反复执行，直到解完一个 slice 层中所有的宏块。decode_macroblock()函数为宏块解码的主体函数，其结构图如图 3-5 所示。

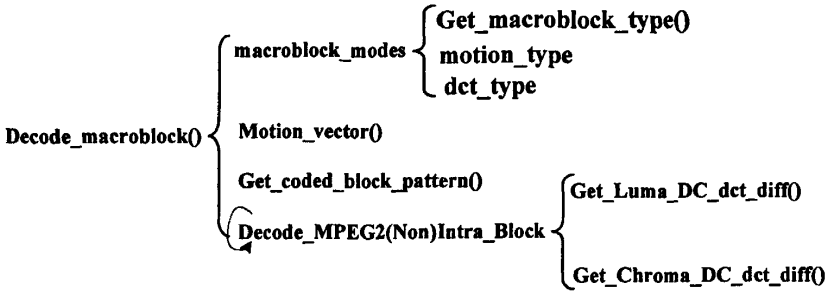

图 3-5 宏块解码结构图

图中，函数 macroblock_modes()用于解出宏块解码所需要的参数，包括：macroblock_type（宏块类型，其值包含：macroblock_intra 表示内部宏块，macroblock_motion_backward 表示补偿的后向预测，macroblock_motion_forward 表示补偿的前向预测）。motion_type 表示预测补偿方式，MC_FIELD 表示场预测，mc_frame 表示帧预测，MC_16X8 表示 16×8 预测类型，仅用于场图，MC_DMV 表示双基预测类型，仅适用于在参考图与被参考图之间没有 B 图的 P 图。这些参数中，macroblock_type 在变长解码时用到，motion_type 在运动补偿时用到，dct_type 在 IDCT 时用到。

函数 motion_vectors()是运动向量解码的主体函数，在这个函数中，解出 PMV 向量，用于运动预测和补偿。

Get_coded_block_pattern()得到参数 coded_block_pattern，用于变长解码。

Decode_MPEG2_(non_)Intra_Block() 表示了两个函数：Decode_MPEG2_Intra_Block( )函数和 Decode_MPEG2_Non_Intra_Block()函数，针对不同的宏块类型，选用不同的函数。之前得到的参数 coded_block_pattern 在这个过程中被用到。针对 4：2：0 的采样结构，对于内部编码的宏块，有 4 个亮度块和两个色度块需要解码，也就是说 Decode_MPEG2_Intra_Block 函数要被调用 6 次，每次针对不同的块进行解码，得到反余弦变换所需的系数。与此同时，反扫描操作也在其中完成。函数 Decode_MPEG2_Non_Intra_Block 针对非内部宏块进行变长码的解码。

解码算法的最后一步是 motion_compensation()执行的，它执行了运动补偿和图像重构的操作，具体的结构如图 3-6 所示。

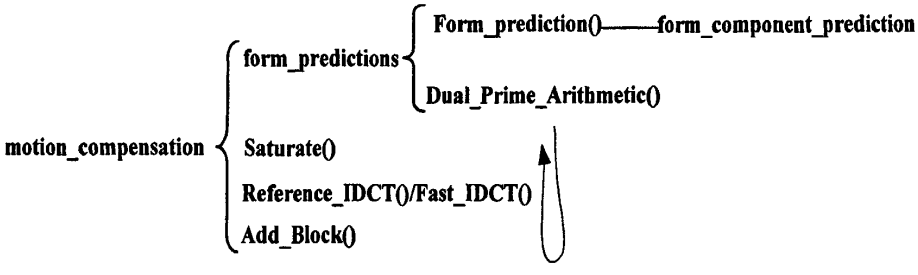

图 3-6 运动补偿&IDCT 结构图

函数 form_predictions()完成了整个宏块的运动预测和补偿两部分的工作，其中包含子函数 form_prediction()和 Dual_Prime_Arithmetic()。每次执行 form_predictions()函数，子函数 form_component_prediction()被调用三次，分别对亮度块和两个色度块进行补偿预测。

函数 Saturate()对反量化的重构结果进行保和化。Reference_IDCT()和 Fast_IDCT()是两种进行反离散余弦变换的函数，作用相同而方法不同，求得的结果相差无几，对实际解码得到的图像几乎没有影响。实际解码时，根据参数 Refernce_IDCT_Flag 在这两种方法中选择其一。最后的 Add_Block()函数对预测值和 IDCT 的结果进行饱和加。这三个函数都是对一个 8×8 的块进行操作的，针对 4：2：0 的采样格式，每个宏块需要执行这组函数 6 次。

3.3 算法的具体实现

在上一节中，根据已有的软件，总结了整个解码系统的函数结构，并剖析出其中实现解码功能的主要函数之间的调用关系，在这一节里，提取函数实现的算法并给与分析。

3.3.1 头信息解码

这部分固定长数据的头信息主要调用 get_hdr()函数来实现。它的主要功能就是从输入流中解码头部信息，直到发现序列结束码或者图像开始码。其解码过程如表 3-1 所示。

表 3-1 头信息解码伪码表

```
#define USER_DATA_START_CODE     0x1B2
#define SEQUENCE_HEADER_CODE     0x1B3
#define SEQUENCE_ERROR_CODE      0x1B4
#define EXTENSION_START_CODE     0x1B5
#define SEQUENCE_END_CODE        0x1B7
#define GROUP_START_CODE         0x1B8

读入数据
if(输入数据==SEQUENCE_HEADER_CODE)   then
  {序列层头数据解码
   扩展数据解码
  }

else if(输入数据==GROUP_START_CODE)  then
  {图组层头数据解码
   扩展数据解码
  }

else if(输入数据==PICTURE_START_CODE)then
  {图像层头数据解码
   扩展数据解码
  }

else if(输入数据==USER_DATA_START_CODE)then
  用户数据解码

else if(输入数据==SEQUENCE_END_CODE) then
  视频解码结束
endif
```

从表 3-1 中，我们清晰得出数据逐层解码的解码方式，但除了各层数据解码

之外，数据码流中还包括各种各样地扩展数据，这些数据一些用来表明图像编码地特征，而一些数据则用在视频解码运算中。扩展数据地解码伪码如表 3-2 所示。

一般来说，函数是循环调用的，所以会解析 sequence_header，GOP_header，直到 picture_header。同时在头信息的解码过程中，包括了对 extension_and_user_data 的解析。

表 3-2 扩展信息解码伪码表

```
#define SEQUENCE_EXTENSION_ID                    1
#define SEQUENCE_DISPLAY_EXTENSION_ID            2
#define QUANT_MATRIX_EXTENSION_ID                3
#define SEQUENCE_SCALABLE_EXTENSION_ID           5
#define PICTURE_DISPLAY_EXTENSION_ID             7
#define PICTURE_CODING_EXTENSION_ID              8
#define PICTURE_SPATIAL_SCALABLE_EXTENSION_ID    9
#define PICTURE_TEMPORAL_SCALABLE_EXTENSION_ID  10

next_start_code()   //获得数据
if(next_start_code==SEQUENCE_EXTENSION_ID) then
  序列扩展解码
else if(next_start_code==SEQUENCE_DISPLAY_EXTENSION_ID) then
  序列显示扩展解码
else if(next_start_code==QUANT_MATRIX_EXTENSION_ID) then
  量化矩阵扩展解码
else if(next_start_code==SEQUENCE_SCALABLE_EXTENSION_ID) then
  序列分级扩展解码
else if(next_start_code==PICTURE_DISPLAY_EXTENSION_ID) then
  图像显示扩展解码
else if(next_start_code==PICTURE_CODING_EXTENSION_ID) then
  图像编码扩展解码
else if(next_start_code==PICTURE_SPATIAL_SCALABLE_EXTENSION_ID) then
  图像时域扩展解码
else if(next_start_code==PICTURE_CODING_EXTENSION_ID) then
  图像空域扩展解码
else
  default
endif
```

3.3.2 可变长解码

MPEG-2 视频数据编码采用的是哈夫曼编码，达到可变字长的最佳编码。在第二章中，我们已经证明，在非均匀符号概率分布的情况下，变长编码总的编码效率要高于等字长编码。此外它的另一优点是，它是一种信息保持型编码，即编解码过程并不引起信息量的损失，因为它的符号和码字之间是唯一对应的。

目前国内外文献中，对哈夫曼解码采用的常用技术有：二元树搜索法[14][15][16]、直接表对应法[17]、分类群组搜寻法[18][19]等。最为传统的哈夫曼解码算法就是二元树搜索法，它是逐位读入数据，然后判断是否存在此码长的码字，如果没有则继续读入一位数据来判断，直到找到正确的码字进行解码。此方法最节省存储空间，但解码效率低下，如果一个码长的字长 12bits 的话，则需要的搜索的次数最少也需要 12 次，这对软件解码来说消耗过多的指令周期，对硬件解码来说耗用过多的时钟周期，所以这种方法的应用场合越来越少。直接表对应法从存储器中查询匹配的字，这种方法的优点是是解码速度快，但需要的

存储空间是最大的（如果最长码字长为 k，则存储空间为 2^k），现在在这种方法的基础上也衍生出其他改进的方法[20]。分类群组搜寻法是二元树搜寻法和直接表对应法的结合，它将查找表划分为几组子表，这样存储空间效率有了较大的改善同时也能获得较快的解码速度。

软件里采用的是 CHT（Canonical Huffman Table）法[21][22]，这种方法利用了解码码表的数据规律：一是具有相同码长的码字是连续增长的；二是对于码长相连的两个码字集合中的码字存在一定的过渡规律，码长小的码字集合中最大的码字加 1 后，再在末尾补充若干个 0 以达到下一个码字集合的位数，这样形成的码字构成了下一个码字集合中的最小码字。利用以上的特点，本算法对查找表进行了重新构造，将码字按字长从小到大排列，根据码长的不同，分化不同的码字集合。为了解码的需要，在集合中还要额外存储一些信息，比如码字的长度等数据。这种算法具有很好的解码性能（存储器利用率高，实现简单等），所以在很多哈夫曼模块的硬件设计中也采取这种方法。

在 MPEG-2 视频标准里，哈夫曼解码包括宏块寻址、宏块类型、宏块模式、运动向量、DC 系数和 AC 系数，其中，AC 系数的编码还用到了游程编码。因为它们解码的原理和算法大都一样，所以我们在这里，以内部块的 DC 系数解码为例，解释软件设计的可变长解码的具体实现方法。在码流中，内部亮度块的编码方式如表 3-3 所示。

表 3-3 亮度块 DC 系数的变长编码

变长码	dct_dc_size_luminance
100	0
00	1
01	2
101	3
110	4
1110	5
1111 0	6
1111 10	7
1111 110	8
1111 1110	9
1111 1111 0	10
1111 1111 10	11

在软件程序中，它把表 3-3 细分成 2 个码字集合：Clumtab0 和 DClumtab1。这两个集合的数据如表 3-4 所示。表中大括号里的第一个数据就是 Huffman 解码的值，第二个数据是表明解码的数据占用的比特数。因为按 CHT 法把数据划分成两个集合，为了解码的方便，又在集合里加了冗余数据。

当程序进行 DC 系数解码时，先从码流中取出 5 位数进行判断，若这个值小于 31，则用表 DClumtab0 来进行运算操作。若对应表中的 3 位有效数据的话，

则低两位无用，而且这两位数据还有“00”，“01”，“10”，“11” 四种情况，所以在集合中{0，3}数据存了四次。其他冗余数据出现的原因都一样。

若取出的 5 位数据大于 31，则编码的数据长度大于 5，则程序直接取 9 位判断，因为此时待解码的数据高 5 位均为“1111 1”，所以取低 4 位数据进行查表判断。因为 4 位数据共存在 16 中情况，所以表 DClumtab1 的深度为 16。

表 3-4 亮度块 DC 系数表

```
/*  dct_dc_size_luminance, codes 00xxx ... 11110 */
  static VLCtab DClumtab0[32] =
  { {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1, 2}, {1, 2},
    {2, 2}, {2, 2}, {2, 2}, {2, 2}, {2, 2}, {2, 2}, {2, 2}, {2, 2},
    {0, 3}, {0, 3}, {0, 3}, {0, 3}, {3, 3}, {3, 3}, {3, 3}, {3, 3},
    {4, 3}, {4, 3}, {4, 3}, {4, 3}, {5, 4}, {5, 4}, {6, 5}, {ERROR, 0}
  };

/*  dct_dc_size_luminance, codes 111110xxx ... 111111111 */
  static VLCtab DClumtab1[16] =
  { {7, 6}, {7, 6}, {7, 6}, {7, 6}, {7, 6}, {7, 6}, {7, 6}, {7, 6},
    {8, 7}, {8, 7}, {8, 7}, {8, 7}, {9, 8}, {9, 8}, {10,9}, {11,9}
  };
```

上面介绍了内部块的 DC 系数解码的软件实现，其余的如色度块 DC 系数、AC 系数等的解码同上述的方法一样，只存在计算复杂度的不同，这里就不再详细介绍了。

3.3.3 反扫描&反量化

在软件程序中，先判断块的类型（一般来说，I 帧图像里所有的块都为内

表 3-5 块数据解码和反量化伪码

```
if(内部块解码)
 {
   decode DC coefficients
   DC系数地反量化
   decode AC coefficients
   /* AC系数反量化地重构  */
     {j = scan[ld1->alternate_scan][i];
      val = (val * ld1->quantizer_scale * qmat[j]) >> 4;
      bp[j] = sign ? -val : val;
     }
   Saturate ()    //饱和化和解谐控制
 }

else if(非内部块解码)
 {
   decode DC coefficients
   DC系数地反量化
   decode AC coefficients
   /* AC系数反量化地重构  */
     {j = scan[ld1->alternate_scan][i];
      val = (val * ld1->quantizer_scale * qmat[j]) >> 5;
      bp[j] = sign ? -val : val;
     }
   Saturate ()    //饱和化和解谐控制
 }
endif
```

部块，P 图中的块可以是内部块，也可以是非内部块，而 B 图中所有的块都为非内部块)。之后，对 DC 和 AC 系数进行哈夫曼解码和游程解码，对获得数据进行反扫描和反量化的重构（依据公式 2-3)。这些任务都在程序中的块解码函数内部完成。表 3-5 的伪码表明块数据的解码过程。

表中，qmat[j]代表量化系数矩阵，quantizer_scale 代表量化步长因子，而 val 则是块解码得出的 ac 系数。另外从程序中可得到软件并没有对 huffman 解码出的数据作反扫描，而是对反量化的计算结果作反扫描，这样操作和标准规定的效果是一样的。

它的重构计算是按照公式 2-3 的绝对化进行，并在计算完成后再作判断。当内部块计算时，公式 2-3 的 k 为 0，所以公式等效于表中的(QF[v][u]*W*quant_scale)/16，非内部块时，k 值就需要作判断。反量化的饱和化过程由函数 Saturate()来实现，计算结果的取值范围为[-2048，2047]，此外系数的解谐控制也在此函数实现。

3.3.4 IDCT 设计

MPEG-2 标准采用了离散余弦变化压缩算法，用以降低视频信号的空间冗余度，其二维公式如公式 2-1 所示。 此时公式中的 N＝8，IDCT 的输入以 12 个 bit 表示，取值范围是[-2048：+2047]，IDCT 的输出用 9 个 bit 表示，取值范围是[-256：+255]。DCT 系数以 12 个 bit 表示，取值范围是[-2048：+2047]。其软件 IDCT 运算的伪码如表 3-6 所示。

表 3-6 IDCT 实现的伪码表

```
/* MPEG-2 IDCT decoder */
 if(Reference_IDCT_Flag==1)
   { Reference_IDCT( )    //采用两次一维矩阵乘法实现
      { idctrow transform()  //一维行变换，矩阵乘法实现
        中间结果的行列变换
        idctcol transform()  //一维列变换，矩阵乘法实现
        结果饱和化
      }
   }

 else
   { Fast_IDCT()
      { for (i=0; i<8; i++)  //一维行变换，快速算法实现
         idctrow(block+8*i);
        for (i=0; i<8; i++)  //一维列变换，快速算法实现
         idctcol(block+i);
      }
     Initialize_Fast_IDCT()  //结果饱和化
   }
 endif
```

而码表中的 Reference_IDCT 函数是直接根据公式计算得到的，虽然比直接二维 IDCT 的计算量大为减少，但每个块还是需要 1024 次乘累加运算，对于软

件解码而言，运算量还是偏大。

所以，自离散余弦变换提出以来，许多研究人员针对 IDCT 运算的理论和算法作出了很多研究，提出了许多不同的快速算法。这些算法主要从算法的计算复杂性的减少和算法结构的简化这两方面着手，以达到提高算法的实现效率的目的。由于在二维 IDCT 运算中存在许多关联的部分，从算法上具有可分解性，所以一个二维 IDCT 可以简化乘两个一维 IDCT 的运算来实现。可以说 IDCT 运算的简化都要统一到一维 IDCT 上来。

```
    /* first stage */
    x8 = W7*(x4+x5);
    x4 = x8 + (W1-W7)*x4;
    x5 = x8 - (W1+W7)*x5;
    x8 = W3*(x6+x7);
    x6 = x8 - (W3-W5)*x6;
    x7 = x8 - (W3+W5)*x7;

    /* second stage */
    x8 = x0 + x1;
    x0 -= x1;
    x1 = W6*(x3+x2);
    x2 = x1 - (W2+W6)*x2;
    x3 = x1 + (W2-W6)*x3;
    x1 = x4 + x6;
    x4 -= x6;
    x6 = x5 + x7;
    x5 -= x7;

    /* third stage */
    x7 = x8 + x3;
    x8 -= x3;
    x3 = x0 + x2;
    x0 -= x2;
    x2 = (181*(x4+x5)+128)>>8;
    x4 = (181*(x4-x5)+128)>>8;

    /* fourth stage */
    blk[0] = (x7+x1)>>8;
    blk[1] = (x3+x2)>>8;
    blk[2] = (x0+x4)>>8;
    blk[3] = (x8+x6)>>8;
    blk[4] = (x8-x6)>>8;
    blk[5] = (x0-x4)>>8;
    blk[6] = (x3-x2)>>8;
    blk[7] = (x7-x1)>>8;
}
```

图 3-7　Chen-Wang 算法的计算程序

1977 年，W.H.Chen、C.R.Smith 和 S.C.Fralick 利用变换矩阵的分解提出了

一种快速算法[23]，采用了非常规则的结构，大大减少计算所需的乘法和加法次数，可以说这是第一个正真的 DCT 快速算法。

Wang 在前者的基础上，于 1984 年提出一种新的算法[24]，其核心思路是先将离散余弦变换转化为离散傅立叶变换（DFT），然后再采用基于离散傅立叶变换的快速计算方法。这种算法不仅适用于 DCT，而且它经过变换，同样可以适用于离散正弦变换等其他的算法，人们一般简称称其为“Chen-Wang 算法”。

除了上述算法之外，研究人员还提出 Lee 算法[25]、Suehiro 算法[26]和 Hou[27]等算法，这些算法的性能大体相同，已达到了运算的极致。

软件程序中调用的 Fast_IDCT 函数就是采用“Chen-Wang 算法”来实现 IDCT 运算的。 如图 3-7 所示，使用该算法，每次一维 IDCT 仅使用 11 次乘法，29 次加法，其总的运算量是 176（11×8×2）次乘法和 464 次加法。比上一节介绍的算法的运算量又有了显著的下降（乘法运算量降低 82.8%，加法降低 54.7%）。由于其算法的出色表现，所以在各种标准视频解码程序中，该算法的使用非常普遍，在本次设计的软件模型中也采用了本算法。

在硬件设计中，用公式计算的方法由于过大的运算量而没用采用。而 Chen-Wang 算法和 Lee 算法的计算性能非常好，但要是硬件实现，其控制电路必将十分的复杂，所以也不采用。由于 Reference_IDCT 函数使用的方法的算法性能和控制实现能很好的折中，所以在具体的硬件电路设计中，采用此算法，具体的硬件实现将在下一章作详细介绍。

在程序中，两种 IDCT 的实现方法都是基于上述原理实现的，分别调用 Reference_IDCT 函数和 Fast_IDCT 函数来实现，两者计算的不同点在于 Reference_IDCT 函数的一维变换采用矩阵相乘的方法实现，而 Fast_IDCT 函数的一维变换采用“Chen-Wang”快速算法实现。

3.3.6 图像重建

当宏块级解码和计算完成后，需要进行图像的重建。Add_Block()函数实现此功能，完成 IDCT 运算结果和预测结果的相加以形成最终的预测块，得到最终的解码样本。

3.4 软硬件设计的不同点

3.4.1 数据的存取

由于软件模型的参考算法是 C 语言代码，故而不需要过于考虑数据存储速度的要求，更无需考虑数据存储的位置，因为对于软件算法而言，它是以整体模块来完成的，只要程序在运行，则定义为全局的数据始终对每一个文件中的模块有效。而在硬件级中，这种全体数据的共享很花费存储空间和存取时间，所以对数据的划分是硬件设计不同于软件设计的首要不同点。这些数据可以分为以下几类：

1．功能参数：这些参数由前一个模块解出而提供给后继模块使用，这些数据量通常较小，并且会被用到的模块也较多，使用频率较高。所以这些数据的解出后，直接存入寄存器组，这样可以大大提高运行速度。

2．变长码解码数据和 IDCT 结果数据：这类数据在每次单独解码时为64bytes，对于变长码解码数据需要被反量化和 IDCT 模块使用，而 IDCT 结果数据则在图像重建时用到。这些数据用到的模块并不多且数据量也不是很大，我们在硬件设计时可以将这些数据单独存储在一个 RAM 中，通过读写 RAM 进行数据的存取。

3．运动补偿的结果数据：这类数据的量相对较大，每一次为 16×16 的块运算，共 256bytes，用法与前一种数据类似，处理方法也类似。

4.图像解码所得原始数据：这部分的数据量很大，在硬件设计中，采用SRAM模块单独来存储。

3.4.2 设计方法

除了数据的实现需要重新设计外，功能模块的划分也是硬件设计的主要任务。在参考软件的 C 语言算法中，各个功能集中在不同函数中，甚至同一个函数可以实现几个功能。这是因为对于纯软件，各个函数可以共用数据，不需要考虑数据流向的因素，程序运行的速度是软件考虑的最重要因素，所以算法的可读性和实现的简洁化才是软件设计非常关心的指标。

而在硬件级别上实现相同的功能则要复杂的多，由于数据不能同时取用，必须将各个功能进行细分，对于可以取用相同部分数据的功能，尽量组合在一个模块或变成相邻模块来实现，对于软件中一个函数可以实现多重功能的情况，可以根据它的数据流向考虑将其细分。由于在硬件设计中，数据是通过总线，由时钟控制进行存取的，因而对数据的存取成为影响硬件运行速度的一个至关重要的因素，尽量节约对数据进行存取的周期数才是实现硬件加速的一个重要途径。

所以对于一个复杂系统，可以通过系统地、反复地将系统划分成较为简单且容易管理和执行的功能单元来完成任务。划分出来地各个功能单元与整体的大单元相比，设计起来更容易，而且测试更简单。对于本文要研究 MPEG-2 视频解码器来说，我们可以把这个大系统根据自顶向下的设计方法，细分成子模块，通过小模块地功能协作来实现设计的目标。

3.4.3 串并行操作

硬件优于软件的最明显的地方就是并行执行的功能结构。对于单 CPU 的计算机，软件执行过程是串行的，尽管可以采用中断等处理方式使空闲的 CPU 执行其他线程，但是严格意义上并不属于几个软件同时运行的状态，因为某一时刻 CPU 处理的只是一个执行行为。而硬件优于软件的一个重要方面就在于此：在时钟的控制下，硬件的每个模块可以同时运行，对于几个没有数据相互依赖

性的模块，可以在控制模块的协调下完全相互独立的执行自己的工作。而对于有数据依赖的模块，虽然前一个模块的数据处理完之后才将数据交给下一级处理，但是，在下一个模块执行时，上级模块又可以进行下一次操作。即使两个模块的执行速度不一致，只要将执行时间调整到两者中的较大值即可。

流水线技术正是体现了这一设计思想，它是一种非常经济且对于提高硬件处理能力非常有效的技术，采用流水线技术可以在不增加硬件或者仅仅需要增加少量硬件就可以将数据处理速度提高好几倍，所以在高速数字电路设计中广泛采用此方法。

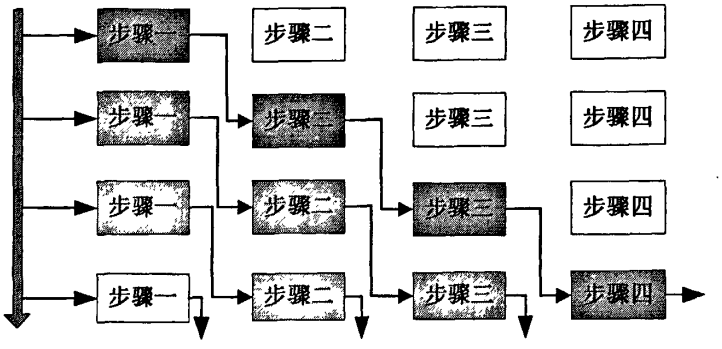

图 3-8 流水线时序图

流水线操作的最大特点和要求是，数据流在各个步骤的处理从时间上看是连续的，如果将每个操作步骤简化假设为通过一个 D 触发器，那么流水线就类似一个移位寄存器组，其技术要点为：

1. 系统分成几个有相互关系的子模块，每个子模块就是一个专门的功能单元。流水线实质上就是将一个大的功能分解成多个独立的小功能来实现，依靠多个功能单元的并行工作来缩短执行程序。

2. 流水线上的个功能单元的后面都需要有数据缓冲区，用于保存本功能单元的处理结果，这是因为各功能单元的处理时间不可能等长，所以需要这个缓冲单元来存储和传送数据。

3. 增加流水线的段数可以提高流水线的吞吐率，但如果段数过多，总的处理时间也会增加，也会增加缓冲区的数据量，所以需要根据处理的需，选择性价比最高的方法来选择流水线的最佳段数。

3.5 硬件设计

上面的章节既介绍了软件的解码过程，也介绍了软硬件设计的区别，这就为我们的硬件设计做好了准备。

图 3-2 所描述的解码流程不仅适用于软件解码，同样也适用于硬件解码。结合参考模型和解码流程，我们可以把硬件解码器划分为以下几个模块：

定长码解码模块(parser)：负责对压缩码流中 slice 层以上的码流解码。

可变长解码模块(VLD)：负责对 slice 层以下，块级以上的码流解码，并解出运

动向量。

哈夫曼解码模块：负责块级数据的可变长解码和完成反扫描任务。

反量化模块(IQ)：通过量化参数重构 DCT 系数。

反离散余弦变换模块(IDCT)：负责对 block 大小的 DCT 系数做逆变换，得到残差数据。

运动补偿模块：根据运动向量找出预测数据，并将预测数据和残差数据相加，得到最终的解码采样值。

本章阐述的这些软件和硬件设计不同点，都将体现在本文的 MPEG-2 视频解码的研究中，具体的硬件解码器的系统架构和模块设计将在下一章作具体介绍。

3.6 本章小结

本章系统地介绍了 MPEG-2 视频解码器地软件实现方法，提取了视频解码的关键函数，给出它们的函数调用关系并作出详细解释。接着，就第二章的关键算法，给出它们软件实现的具体过程。阐述了软硬件设计的不同点，描述了硬件解码器的大体结构，为下一章硬件设计指明设计目标。

第四章　MPEG-2 视频解码器的硬件实现

4.1 MPEG-2 视频解码器的总体架构

在前面章节中，已经对 MPEG-2 解码算法和关键技术进行了详细的介绍，本章主要阐述解码器的硬件设计方法以及在结构上的实现，整个 MPEG 解码器的架构和模块划分如图 4-1 所示。

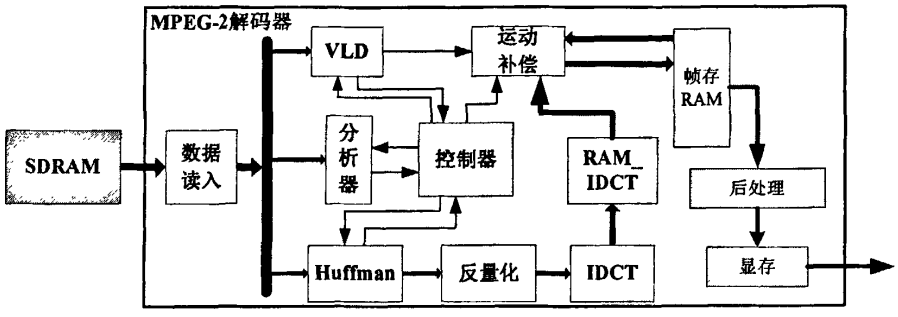

图 4-10 解码器的总体架构图

如上图所示，本文设计的 MPEG-2 视频解码器主要分为 8 个模块，读入模块（regdata）、parser 模块、功能控制模块、可变长解码、huffman 模块、反扫描反量化模块、IDCT 模块、运动补偿模块和后处理模块。在下面的章节里将介绍各个功能单元的实现方法以及各个模块间的工作关系。

4.2 读入模块设计

本模块从传输的码流中读取一定位数的编码数据，经过桶式移位器的处理后，输出特定位数的数据供解码使用。

4.2.1 设计实现

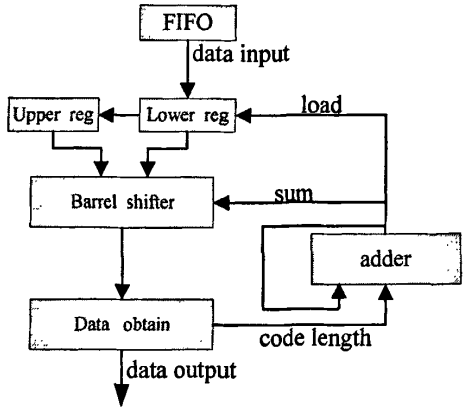

图 4-2 读入模块的功能实现图

在系统设计中，一次解码所需要的数据位数最大为 32 位，所以使用两个 32

位变量的寄存器的组合来提供解码所需要的数据（桶式移位器的输入数据 64 位），就是图 4-2 所示的 upper reg 和 lower reg，寄存器 upper reg 的数据是通过 lower reg 寄存得到的。

一开始，先通过 lower reg 从 FIFO 读入数据送给 upper reg，接着 lower reg 再次读入数据覆盖原来的数据，至此所有的输入数据已送至桶式移位器的输入端。

解码一次需要的位数通过 code length 送入加法器累加得到 sum 信号，桶式移位器通过对 sum 数据的判断来决定桶内数据的最高位，并从这最高位开始截取 32 位数据输出。当加法器累加的数据超过 32 时，load 信号有效表明 upper reg 中的数据已经被完全解码过，此时循环上面所述的操作，这样就可以数据读入的正确性和连续性。下面已具体实例来解释上面描述的计算过程。

4.2.2 RTL 级仿真结果

模块级别的仿真波形如图 4-3 所示，其中的输入数据是从 FIFO 读入，送入 upper reg 和 lower reg。length 表示桶式移位器最高为移出的位数。distance 表示已移出位数的累加和，若值达到 32，则变为 0，输出一个时钟周期高脉冲的 load 信号，此时 doe 信号变为低电平，移位器的输出数据不能正确使用。

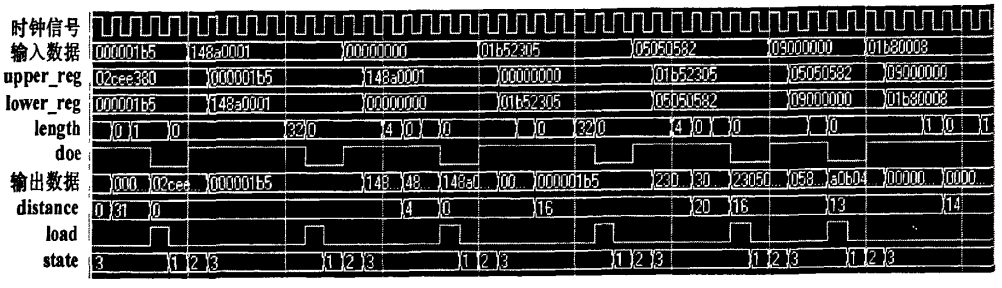

图 4-3 读入模块仿真波形

从图中，我们还可以得出，实现上述操作是由一个小状态机实现的：

状态 1：读 FIFO，把 lower reg 里的数据写入 upper reg。

状态 2：把 FIFO 里的数据写入 lower reg。

状态 3：状态保持，输出有效数据，寄存给后继模块使用，当 load 信号为高时，跳回状态 1 开始循环操作。

4.3 功能控制模块设计

视频解码是包含了多个任务的集合，各个单元在运算复杂度、密集度等方面有很大差异，根据不同的算法特性可以设计合理的功能单元，如 VLD、IDCT 模块以符合硬件设计的要求。这些子模块各自独立地负责实现相应的算法。但要完成视频解码则需要一个整体控制结构，负责将各子模块连接起来，使之相互协调地配合工作。本模块(top_state)就是完成此功能，协调视频解码的每一步骤，以此来完成整个视频的解码和显示工作。这个模块的设计思想就是整个解

码器的设计思想，它决定了各个模块的工作方式和解码效率。

4.3.1 解码控制方法

本模块采用集中控制策略[28][29]，模块在数据驱动下工作，当输入数据有效，模块工作，当输入数据无效，模块暂停工作。

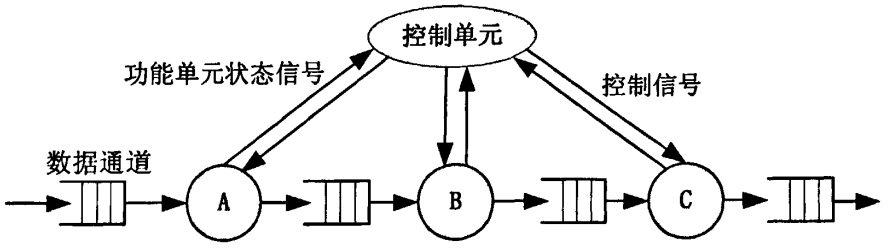

图 4-4 控制策略图

图中所示的 A、B、C 为功能的单元的抽象表示，功能单元间相互传送的仅仅是处理的数据，功能单元与控制单元间传送的是状态信息。控制单元和功能单元传送的是控制信息。在这种控制方式下，由中央控制单元来控制各功能单元的运行和停止。在具体的设计时，此模块是根据各个模块的信号产生的交互信号（功能单元状态信号）作出仲裁，发出控制信号，将各个子模块有机的结合起来，构成一个完成的系统。其模块部分输入输出信号如表 4-1 所示。

表 4-1 控制模块的部分接口信号

信号名称	输入/出	作用
go	输入	整个系统的启动信号
parser_done	输入	帧头分析器模块结束信号
vld_done	输入	可变长解码模块的结束信号
mc_done	输入	运动补偿的结束信号
huffman_done	输出	Huffman 解码模块结束信号
parser_go	输出	帧头分析器模块启动信号
vld_go	输出	可变长解码模块启动信号
Huffman_go	输出	Huffman 解码模块启动信号
mc_go_skipped_mb	输出	运动补偿启动信号之一

解码器启动时，控制模块最先发出 parser_go 信号，启动分析器模块，占用数据读入模块的输出，进行固定长数据的头解码和冗余数据的舍弃，当工作完成后，给 top_state 发回一个返回信号 parser_done， top_state 模块此时作出判断：当图像为 I 帧图时，给出 VLD 模块的启动信号（vld_go 置高电平），占用数据读入模块的输出，VLD 模块工作完成后同样发回一个返回信号 vld_done 给 top_state。此时，huffman 模块启动，数据读入模块的输出供其使用，开始块级的数据可变长解码，因为解码的视频格式为 4：2：0，所以一个宏块内最多含有 6 个块，在每个编码块解码结束时，启动反量化和反 DCT 模块（这两个模块的启动不需要经过 top_state 模块）。当整个宏块解码完成，控制模块重新

按照上述顺序进行循环操作；当解码图像为 P 帧或 B 帧时，判断宏块是否产生跳跃，若没有宏块跳跃，则模块的启动方式同 I 图一样，只不过在帧内模块工作的同时，运动补偿模块同时进行工作，当 IDCT 模块停止工作时，就进行图像的重建。

4.3.2 设计实现

模块具体的功能实现主要用状态机来完成。其状态转移图如图 4-4 所示。

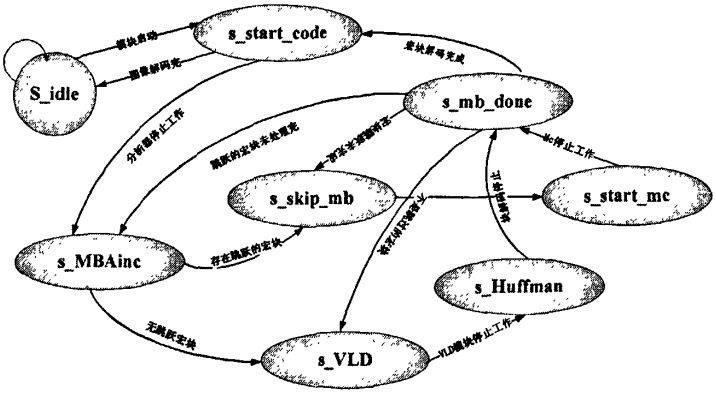

图 4-5　控制模块状态转移图

s_idle 状态：空闲状态。当启动信号有效时，进入 s_start_code 状态开始工作。

s_start_code 状态：标志此时 parser 模块处于工作状态。如果 paser 模块的结束信号有效时（高电平），跳入 s_MBAinc 状态。

s_MBAinc 状态：在此状态获得信号 macroblock_address_increment 的值，以此信号判断是否产生宏块跳跃。若宏块之间存在跳跃，则跳入 s_skip_mb 状态，否则进入 s_vld 状态。

s_skip_mb 状态：根据宏块的跳跃数目，获得当前宏块的地址，进入 s_start_mc 状态。

s_vld 状态：标志 VLD 模块处于工作状态，正在进行到块层的可变长数据解码，并计算出宏块的运动向量。

s_huffman 状态：标志块层数据的可变长数据解码和计算正在工作。

s_start_mc 状态：标志运动补偿模块开始工作。在本状态等到 IDCT 工作完成后，进行宏块级别的图象重建，完成跳入 s_mb_done 状态。

s_mb_done 状态：在本状态 MBAinc 信号进行减一操作，之后根据信号 MBAinc 数值的大小，决定不同的状态跳转，只用于 P 图、B 图。若没有宏块跳转，则 MBAinc 减一后为 0，则跳入 s_start_code 状态，相当于完成一个宏块的处理。若 MBAinc 减一后其值大于 1，则继续有宏块需要跳过。若 MBAinc 减一后其值等于 1，则此地址的宏块就是被跳过的宏块的最后一个，需要进行处理，所以进入 s_vld 状态，开始解码。

4.4 Parser 模块设计

4.4.1 实现原理

在实际的传输过程中，被传送的是经过压缩的视频数据码流。这个码流也是按照一定的语法经过编码而成，这个语法称之为视频序列。所以此模块的主要任务就是根据它的编码语法还原出需要的数据，舍弃无用数据。

因为视频序列的存在，我们从 FIFO 读入的数据是有规律可循的。这就使得我们硬件解码就有实现的可能[30][31]。

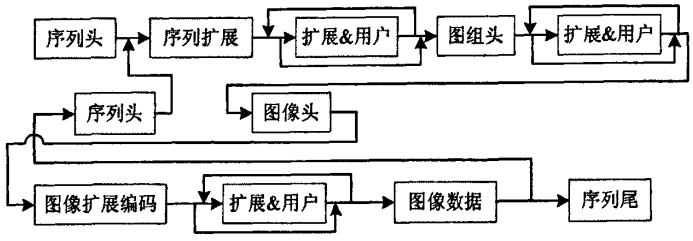

图 4-6 MPEG-2 码流组织结构

本模块工作过程如下：由 FIFO 读出数据送入桶式移位器，桶式移位器的输出作为解码使用的数据，首先判断数据是否为块数据，若是则此模块停止工作，否则本模块继续工作，即起始码的识别，然后看起始码后是否跟有扩展码，按照序列头，序列扩展，图组头，图像头，图像编码扩展，片头，宏快数据的顺序依次识别。解码过程中产生的有用数据送至参数寄存器，以供系统解码中的各个功能模块的调用，参数寄存器中的数据随解码过程而不断刷新。使用桶式移位器进行移位，累加器的输出控制桶式移位器的移位位数，当累加器的进位输出为 1 时，表示桶式移位器的输入 reg B 中的数据全部移完，此时 REG A 中的数据移植 REGB，FIFO 读出数据送入 REG A，这样解码过程就不会中断。

4.4.2 实现的功能

通过分析器模块与其它模块之间的数据关系，结合 MPEG-2 标准，设定分析器模块实现的主要功能如下：

1. 帧头的固定长数据解码：对数据读入模块输出的数据进行解码，包括提取、舍弃、运算等操作。

2. 色彩转换模块控制实现：分析器的状态机工作到特定状态（当检测到图像头时），分析器暂停工作，启动一个 YUVtoRGB 模块，图形开始以帧为单位进行上采样和颜色的空间转换，通过显示策略，把图像显示出来。当 YUVtoRGB 模块工作完成后，分析器模块继续解码工作，直到宏块片层，停止工作。此外，因为硬件解码的速度过快，远超过图像的显示速率，所以这个功能的实现也保证了解码帧速率的控制。

4.4.3 控制实现

设计采用有限状态机来实现分析器的解码工作。使用的状态转移图如图4-7描述。由图可以看出，此状态机根据MPEG-2视频码流的层次性特点出发，同时结合解码的系统结构，增加部分状态来实现既定的功能。其主要状态有：

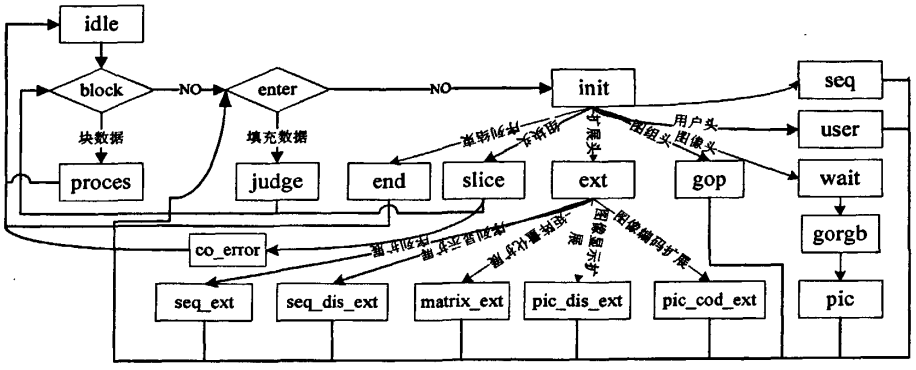

图4-7 解码控制状态转移图

idle状态：空闲状态。当启动信号有效时，进入block状态开始工作。

enter状态：检查桶式移位器的高24位输出是否为24`h000001，确定起始码前缀；

init状态：判断状态，在这一状态检查起始码，根据不同的起始码进入不同的状态，如桶式移位器输出的值为32`h000001b2，则跳入user状态，进行用户数据的处理，如起始码值为扩展码32`h000001b5，则需对数据再做判断，以决定进行哪一种扩展解码；

头信息解码状态：包括seq、user、gop、pic、slice、和一些扩展状态，用于解码相对应的码流序列数据。

除了上述主要解码状态，此状态机还根据系统要求设置了其他一些状态，用于执行一些特定功能：

block状态：检查码流中数据的是否为块数据，如桶式移位器输出为块数据，则本模块停止工作；

judge状态：表示输出的最高位数据多余，移出1位不用；

wait状态：因为用硬件解码速度率高于视频标准速率要求，所以在解码过程中间插入等待状态，当解到图像头时，停止解码；

proces状态：表示桶式移位器输出数据是宏块或块数据，给出一个信号，表示本分析器停止工作，桶式移位器的输出数据交由其他模块处理；

gorgb状态：启动YUVtoRGB模块的工作，此模块实现上采样、颜色空间的转换和显示功能。当此模块工作完成后，状态机跳入pic状态，开始图像头的解码，实际上此后继模块的工作对象是上一帧解码出来的数据。

co_error状态：纠错状态[32]，分析器只解码到组块层，而组块又是MPEG2中的基本容错单元，所以我们在这里设定此状态用于纠错，如果不作纠错处理，

可能因为码流中的语法元素的误码而造成解码错误（由于起始码异常造成显示位置的错乱、组块中因为误码而无法解出的宏块），通过纠错可以最大限度地利用了接收到的码流，也尽可能地保护了后续码流解码地正确性。

4.4.4 仿真结果

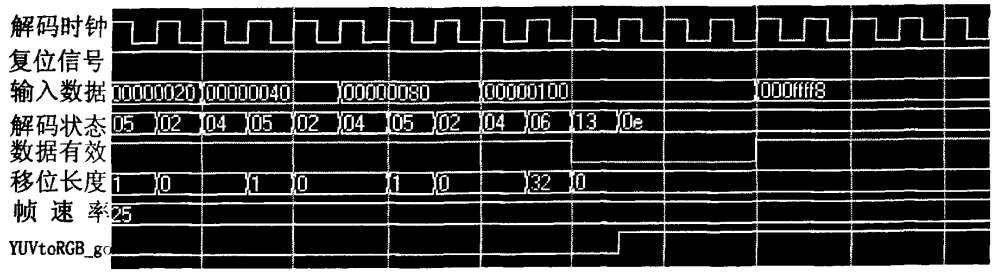

图 4-8 分析器仿真部分结果图

输入数据表示的是从桶式移位器截取的高 32 位数据作为本模块的输入，数据有效信号高电平时表示输入数据是可解码的，若电平为低则不能用于解码，解码状态信号表示此时主状态机的状态。图中‘06’表示解码的是标志状态极正在对起始码进行判断。移位长度信号表示解码多少数据，桶式移位器需移出的数据量。帧速率信号表示解码图像 1 秒显示 25 帧。

4.5 VLD 模块设计

4.5.1 设计功能及原理

在本模块中，主要完成到块级的可变长解码，块级的可变长解码由

表 4-2 宏块编码方式表

macroblock(){	位数	助记符
while(nextbits()=='0000 0001 000')		
macroblock_escape	11	bslbf
macroblock_address_increment	1~11	vlclbf
macroblock_modes()		
if (macroblock_quant)		
quantiser_scale_code	5	uimsbf
if (macroblock_motion_forward ||		
(macroblock_intra && concealment_motion_vectors))		
motion_vectors(0)		
if (macroblock_motion_backward)		
motion_vectors(1)		
if (macroblock_intra && concealment_motion_vectors)		
marker_bit	1	bslbf
if (macroblock_pattern)		
coded block_pattern()		
for(i=0;i<block_count;i++){		
block(i)		
}		
}		

huffman 模块来完成。此外它还完成宏块的运动向量的计算过程。

按照标准中定义的宏块的内容进行解码操作，如表 4-2 所示。其中表中的数据 macroblock_escape 和 macroblock_address_increment 已经在 top_state 模块里已经进行过解码了，所以本模块的可变长解码就是从 macroblock_modes 开始。macroblock_modes 的具体编码内容随着编码图像的不同而改变。

4.5.2 设计实现

在模块设计中[33][34]，顶层控制采用状态机来实现解码。状态机的状态转移图如下图所示。

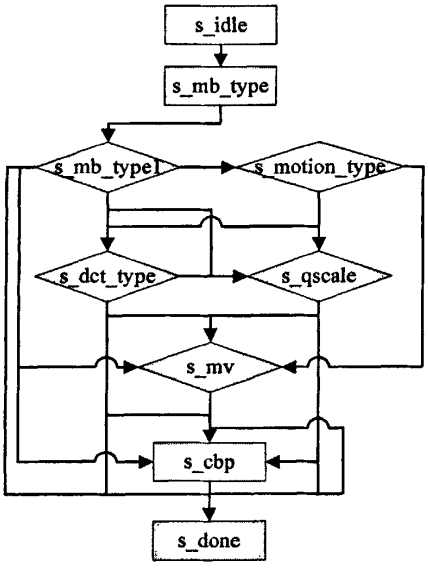

图 4-9 可变长解码状态转移图

s_idle 状态：空闲状态，模块启动时，跳入 s_mb_type 状态。

s_mb_type 状态：根据可变长数据 macroblock_type 进行解码，译码得出一定量的控制信号，用于后继模块的运算。

s_mb_type1 状态：判断状态，

s_dct_type 状态：如进入本状态，根据 decode_dct_type 判断出宏块是帧 DCT 编码还是场 DCT 编码的，由 dct_type 信号表示。decode_dct_type 的获得方法如下：

```
if((picture_structure==“frame”)&&(frame_pred_frame_dct==0)&&
  (macroblock_intra||macroblock_pattern)) {
        decode_dct_type=1; }
else
      decode_dct_type=0;
```

本状态占用 1 个时钟周期，表示宏块模式的解码完成，判断进入其他状态。

s_qscale 状态：获取 quantizer_scale_code 信号，判断进入其余状态。

s_mv 状态：启动 mv_vld 模块，完成可变长解码和运动向量的计算，完成跳入

s_cbp 状态或 s_done 状态。

s_cbp 状态：cbp 变量的计算，因为图像格式为 4：2：0，所以最大值为 63。

s_done 状态：结束状态，表示整个模块工作的完成，VLD 模块停止工作，之后数几个周期运动补偿模块开始工作。

4.5.3 运动向量解码

因为运动向量是进行差分编码的，所以当前的运动向量主要由两部分组成的，即运动向量差值（delta）和运动向量 PMV。运动向量解码结构图如下所示。

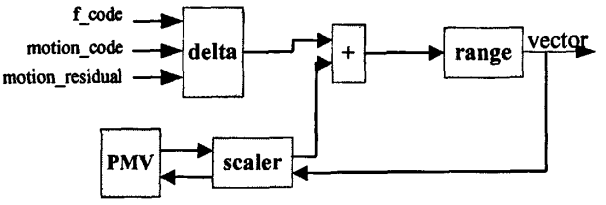

图 4-10 运动向量解码结构图

在图中，f_code 为运动向量解码时的一个参数，该参数只出现在预测图像中，它决定着被解码向量的最大值，其值范围为 1 至 9。motion_code 和 motion_residual 分别为标志向量值和向量误差，PMV 为前一个运动向量值。vector 为重构后的运动向量。

其解码过程是：通过可变长解码获得的 f_code、motion_code 和 motion_residual 进行计算得到 delta，这就是上一个运动向量与当前运动向量的差值。同时 PMV 送出上一个运动向量值，经过 scaler 处理后与运动向量差值相加，再经过饱和化后即可得到当前要解码的运动向量。为了便于下一次的运动向量解码，解码后的向量需再经过 scaler 处理后，送回 PMV 寄存。

1．delta 计算

实现的算法如下：

$$|delta|=\begin{cases}|motion\_code| & (f\_code=1 \quad or \quad motion\_code=0)\\ (|motion\_code|-1)\times 2^{f\_code-1}+|motion\_residual|+1\end{cases} \qquad (4\text{-}1)$$

式 4-1 中 2 的 N 次方可用移位寄存器来实现。motion_code 是可变长编码的，且可正可负。在本模块设计中，负数用补码表示。所以 motion_code 的绝对值可以通过对最高位的符号位进行判断来实现。

2．PMV 单元是一个寄存器组

运动向量解码得到的最终要再寄存到 PMV 中，为下一次向量解码做准备，在下面情况下，PMV 要复位：

(1) 在每个组块的起始处。

(2) 在没有隐藏运动向量的内部宏块被解码时。

(3) 在 P 图中，当 macroblock_motion_forward 为零的非内部宏块被解码时。

(4) 在 P 图中，当一个宏块被跳过时，即 macroblock_address_increment>1。

3. Scaler 处理

MPEG-2 中为了尽量提高压缩效果，允许运动补偿有许多种：针对帧图像的帧预测、场预测及双基预测；针对场图像的场预测和双基预测等不同组合。当运动补偿方式是针对帧图像的场预测时，scaler 对输入的运动向量进行乘以 2（饱和化后的运动向量作为输入时）或除以 2（PMV 单元中的运动向量作为输入时），当其他运动补偿方式时，scaler 对输入点运动向量不进行操作。

4. Range 处理

主要是对得到的运动向量进行饱和化，他的算法如下：

$$vector=\begin{cases} delta+pmv+32*2^{f\_code-1} & (delta+pmv<-16*2^{f\_code-1}) \\ delta+pmv-32*2^{f\_code-1} & (delta+pmv\geqslant -16*2^{f\_code-1}) \\ delta+pmv \end{cases} \quad (4\text{-}2)$$

式 4-2 的功能是把 vector 的值限定在（delta + pmv - $16*2^{f\_code-1}$～delta + pmv + $16*2^{f\_code-1}$）之间。

4．6 反量化

本模块为纯计算单元，其计算原理在第二章已经具体介绍过，本节主要研究它的硬件设计方法。很显然，在反量化的过程中，要用到两个矩阵，对 4:2:0 格式的数据，一个用于内部宏块，另一个用于非内部宏块。在解码中，矩阵有可能是默认的，也有可能是自定义的，即从码流中解出新的矩阵。为了解码方便和便于控制，在设计中，量化比例因子以组合逻辑的形式存在，而自定义的矩阵则存在 RAM 中。

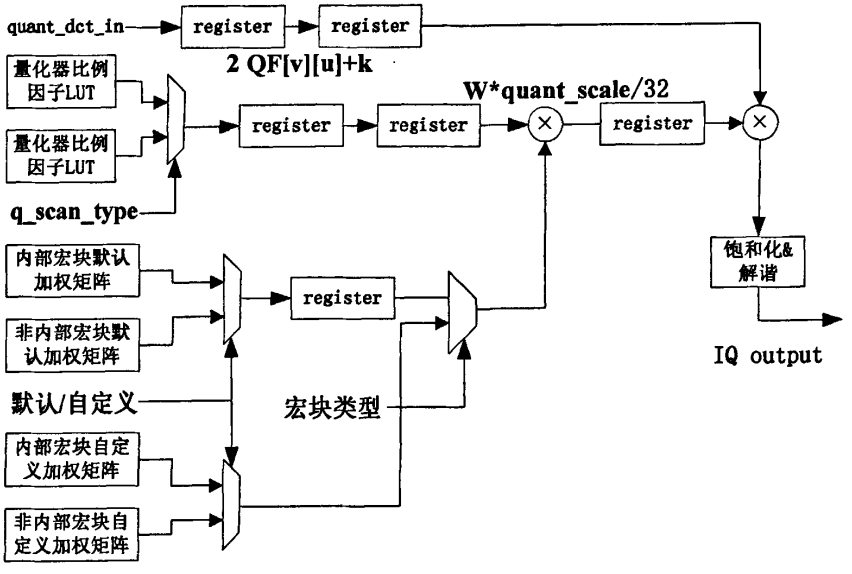

图 4-11 IQ 电路结构图

为了体现硬件的设计思想，本模块内部数据处理采用流水运算，大大减少了

数据处理时间[35]。设计时需要研究的问题如下：

1. 系数首先要取绝对值，其次对解码所得的系数进行相应左移位，再对块的类型（内部块/非内部块）、系数的正负作判断，决定左移位的系数绝对值是否加 1 运算。

2. 时序的给定，考虑数据的计算过程，在特定的时序节拍下给出特定的数据。

3. 数据输出的字长，它与最后所需的精度有关。在设计中，量化的最后输出是 12 位的 DCT 系数，而 qscale/32 * Qmatrix 计算的结果为 15 位数据，而 2*quantised input + k 计算结果为 12 位数据，所以反量化重构后的结果有 27 位数据，经过饱和化后，其有效数据位只有 12 位，也是我们最终输出 DCT 系数的位数。

在程序中，步长因子表中的数据以 quantiser_scale×8 代替标准定义的数据。

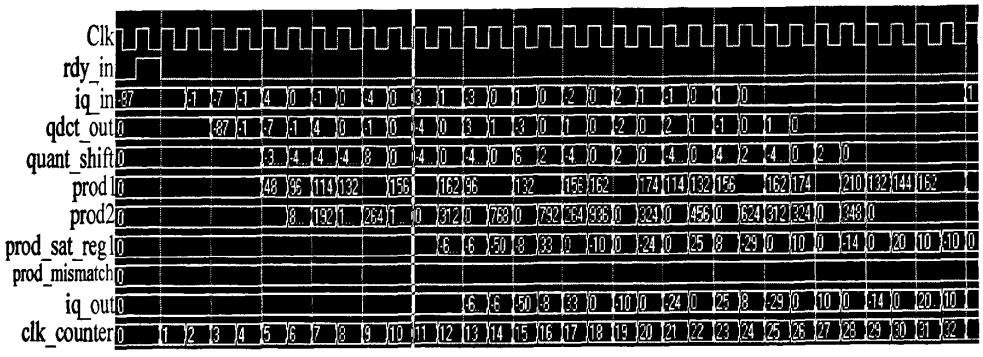

图 4-12 IQ 模块仿真波形图

由仿真图形可得，rdy_in 信号表示模块启动，qdct_out 对输入数据进行两级寄存，间隔 3 个时钟周期，获得输入数据的 2QF[v][u] + k 值 quant_shift 和 qscale/32 ×Qmatrix 值 prod1。而在间隔的第 4 个时钟周期，通过乘法器获得（2QF[v][u] + k）×qscale/32×Qmatrix 的计算结果 prod2。在间隔 11 个时钟周期之后，prod2 通过饱和化过程，得到数据 prod_sat_reg1。寄存一级，判断是否要解谐控制，若没有，则输出反量化的结果。可见需经过 12 个 clk，输出第一个值，以后每一时钟输出一个值。所以完成一个块的反量化过程最多需 75 个时钟周期。iq_out 是模块的数据输出信号，送入 IDCT 模块，clk_counter 是时钟计数器。

4.7 IDCT 模块

在我们设计的 MPEG-2 视频解码器中，用到的是大小为 8×8 的图像块，如果直接进行二维 IDCT，就要进行 8192 次乘法和 3548 次加法操作，这就导致控制电路的复杂，且运算效率的低下，具体表现就是占用过长的时钟周期。所以在实际的硬件电路设计中，往往对算法进行优化[36][37]，来提高整个 IDCT 的运算速度。在第三章介绍了两种快速算法，并比较了它们的优劣，选取一种算法作为硬件设计的算法，其具体设计过程如下所述。

4.7.1 设计原理

二维 8×8IDCT 变换定义为：

$$x(i,j)=\frac{1}{4}\sum_{i=0}^{7}\sum_{j=0}^{7}C(u)C(v)X(u,v)\cos\frac{(2i+1)u\pi}{16}\cos\frac{(2j+1)v\pi}{16} \qquad (4\text{-}3)$$

其中，x（i，j）是象素值，X(u，v)是编码变换后的系数。

之所以二维 IDCT 运算采用行列分解的结构来实现。首先是因为大多数的直接二维 IDCT 算法结构往往很不规整，大量使用全局互联，不适合采用 FPGA 或 ASIC 电路来实现；另外一个重要的原因在于基于现有的一维 IDCT 结构来实现二维 IDCT 比重新设计一个新的二维 IDCT 结构更加有吸引力，这样可以节约大量的设计时间。所以，在本文中 IDCT 设计采用行列分解的方法来实现二维 IDCT 运算，并用矩阵相乘的方法实现行或列的一维 IDCT 运算，这和软件模型中 Reference_IDCT 函数所用的算法相一致。

则公式 4-3 也可用矩阵形式表示[38]：$f=G^{T}\bullet F\bullet G$，其中

$$G=k\bullet\cos\frac{(2\cdot col\_num+1)\bullet row\_num\bullet\pi}{2M} \qquad (4\text{-}4)$$

$k=\sqrt{\frac{1}{8}}$，当 row=0。$k=\frac{1}{2}$，当 row≠0。

式 4—4 就是一维 IDCT 的行计算公式。得到具体矩阵如下所示。

$$G=\begin{pmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \cdots\cdots & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cos\frac{\pi}{16} & \frac{1}{2}\cos\frac{3\pi}{16} & \cdots\cdots & \frac{1}{2}\cos\frac{15\pi}{16} \\ \cdots & \cdots & & \cdots \\ \cdot\cdot & \cdot\cdot & & \cdot\cdot \\ \frac{1}{2}\cos\frac{7\pi}{16} & \frac{1}{2}\cos\frac{21\pi}{16} & \cdots\cdots & \frac{1}{2}\cos\frac{105\pi}{16} \end{pmatrix}$$

，转换为定点整数为(相当于把上一个矩阵里的值乘以 256)：

$$G=\begin{pmatrix} 91 & 91 & 91 & \cdots\cdots & 91 \\ 126 & 106 & 71 & \cdots\cdots & -126 \\ 118 & 49 & -49 & \cdots\cdots & 118 \\ 106 & -25 & -126 & \cdots\cdots & -106 \\ 91 & -91 & -91 & \cdots\cdots & 91 \\ 71 & -126 & 25 & \cdots\cdots & -126 \\ 49 & -118 & 118 & \cdots\cdots & 49 \\ 25 & -71 & 106 & \cdots\cdots & -25 \end{pmatrix}$$

4.7.2 一维 IDCT 设计

二维 IDCT 被行列分解为两个一维 IDCT，图 4-13 所示。首先进行的是行变换，等到第一行的 8 各个数据全部到齐之后，送入一维 IDCT 处理单元，经过 8 个时钟周期处理之后，然后将数据写入转置存储器，将输入数据换序后输出，

在设计中用一个空间为 64 的 RAM 来实现，开始第二行的一维 IDCT 运算。在全部 8 个行 DCT 变换计算完成后，开始列变换。

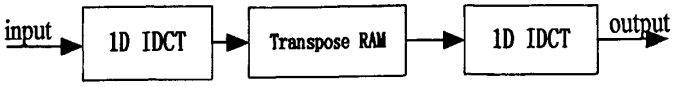

图 4-13 二维实现结构

计算过程如下，已知 $f = G^T \bullet F \bullet G$，也可以写成 $f = G^T \bullet Z$，而 Z=F・G。按照矩阵相乘的运算法则， F 矩阵的第一行的每一个元素要和矩阵 G 的第一列元素相乘加才能获得过渡矩阵一个值 Z_{00}，同样 F 矩阵的第一行的每一个元素要和矩阵 G 的第二列元素相乘加才能获得过渡矩阵一个值 Z_{01}，当 F 矩阵的行数从 0 计算到 7 时，当 G 矩阵的列数从 0 计算到 7 时，按照上面的计算过程，则 8×8 的 Z 矩阵就可计算出来。图 4-14 所示的一维 IDCT 计算的结构图。

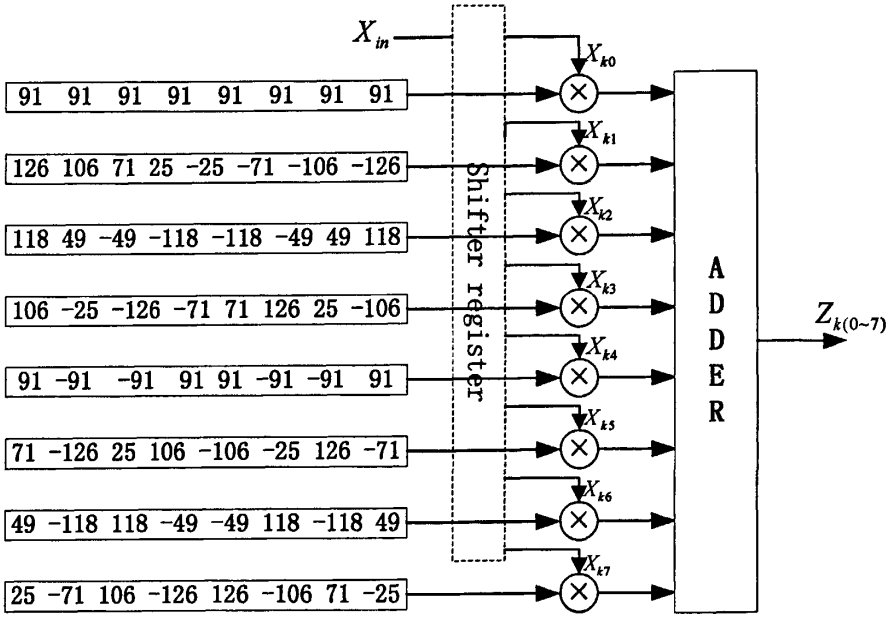

图 4-14 一维 IDCT 结构图

由图 4-14 可知，计算一个值 Z_{00} 需 8 次乘法操作和 7 次累加操作，所以一次一维 IDCT 的计算需 512（64×8）次乘法和 448 次加法，则完成二维 IDCT 的需要 1024 次乘法操作和 896 次加法操作。比直接进行二维 IDCT 的 8192 次乘法和 3548 次加法操作的运算量大为下降。

在实际的电路设计中，矩阵 G 中的 64 个固定系数被分为 8 个寄存器组存储，共 8 组，每一组 8 个寄存器共存储 G 矩阵的一列数据，视具体情况选择哪一列数据进行运算，比如计算一开始时，选择 G 矩阵的第一列数据，接下来选择第二组，这样选择下去就能实现图 4-14 所示的移位寄存器的功能。具体设计的电路图如图 4-15 所示。

在电路图中，输入数据被寄存 8 次，第一个原因是通过寄存数据的改变来实

现图 4-14 所示的移位寄存器的功能。第二个原因是为了实现计算的流水操作。当 8 位数据输入计算单元时，选择第一组寄存器的数据和输入数据进行程累加，直到 8 组寄存器选择完成。当计算完毕时，8 个输入数据寄存器已被新的数据覆盖，循环上述运算可以实现流水运算，大大提高了计算效率。

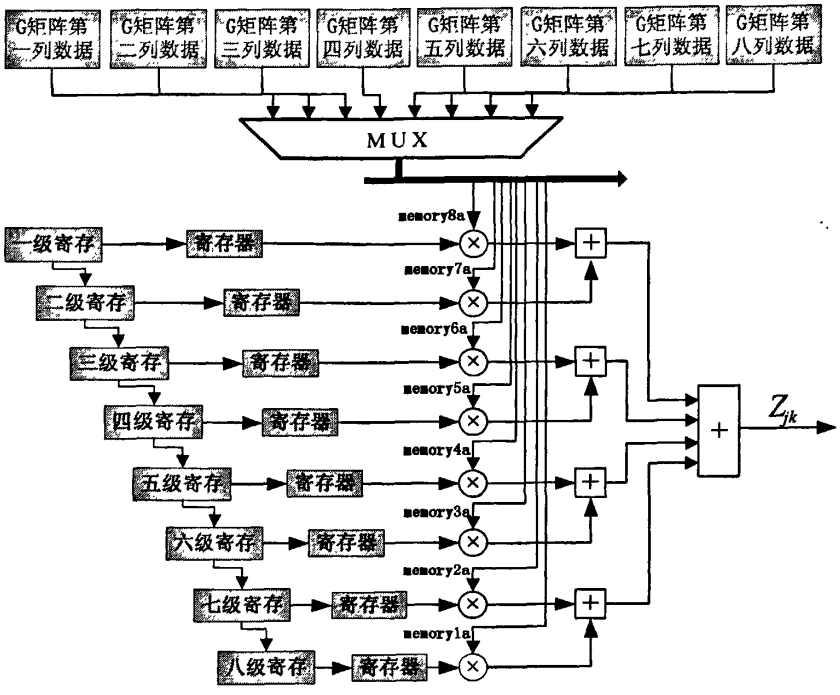

图 4-15 一维 IDCT 电路实现图

4.7.3 转置 RAM

转置 RAM 可以形象地看成是 8×8 序列，存储的前 8 个字节对应着阵列的第一行，第 9 到 16 个字节数据对应着阵列的第二行，依次类推，需要的 RAM 空间大小位 64bytes。

表 4-3 转置 RAM 写顺序

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

表 4-4 转置 RAM 读顺序

1	9	17	25	33	41	49	57
2	10	18	26	34	42	50	58
3	11	19	27	35	43	51	59
4	12	20	28	36	44	52	60
5	13	21	29	37	45	53	61
6	14	22	30	38	46	54	62
7	15	23	31	39	47	55	63
8	16	24	32	40	48	56	64

如果一维 IDCT 计算出地中间结果按表 4-3 的顺序存储，则列变换时按表 4-4 的顺序读取数据。如果一维 IDCT 计算出地中间结果按表 4-3 的顺

序存储，则列变换时按表 4-4 的顺序读取数据。

4.7.4 RTL 级仿真波形

程序通过仿真，得到部分波形如下图所示。当启动信号有效（高电平时）时，模块处于工作状态，开始输入反量化模块的计算结果，当输入数据移位器的 8 个输出被寄存，此时可以和系数矩阵的列数据进行乘法操作。把得到的 8 个乘法运算的结果经过 3 个时钟周期作累加计算，所以输出第一次一维 IDCT 的结果需 13 个时钟周期。因为内部采用流水的计算方法，所以以后每个 clk 都输出一个结果存入转置 RAM 里去。当所有的中间值都存完以后，开始从转置 RAM 里读出数据，进行第二次的一维 IDCT 变换。这个过程与上一次转换几乎一样。波形如图 4-16 所示。

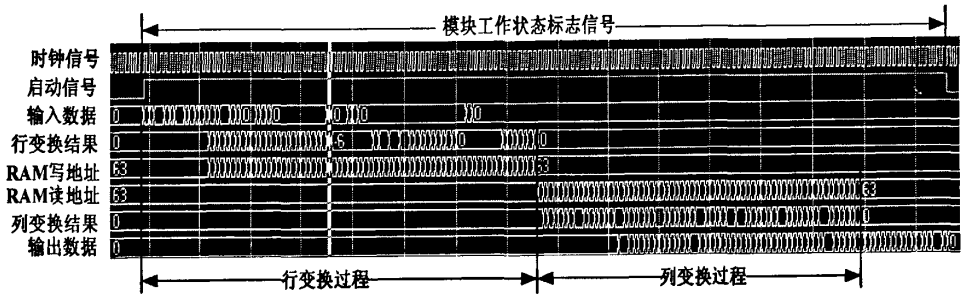

图 4-16　IDCT 模块仿真波形图

可见，完成一个块的 IDCT 运算需要 156 个时钟周期，其中行列变换各占一半的时间，因为转置 RAM 数据的读取时间隐藏在数据运算的过程中，所以这一操作并不消耗整个模块的运行时间，所以在这里没有必要把 RAM 用寄存器替换。

4.8 运动补偿模块

运动补偿是动态图像编解码的特有技术，是视频解码的最为关键一环。因为运动补偿模块的 VLSI 实现牵涉到大量的运算和频繁的数据存取操作。所以在设计中[39]，把这个模块划分为几个子模块，这样较容易实现和理解。

本文中的运动补偿电路如图 4-17 所示。

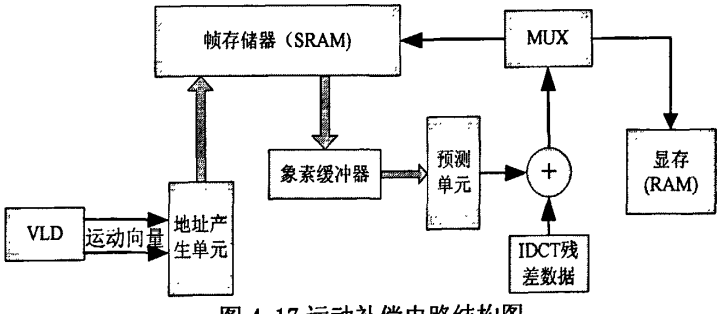

图 4-17 运动补偿电路结构图

4.8.1 地址产生单元

该单元主要产生当前帧的宏块地址和预测帧的宏块地址。对于内部宏块，不用预测，只需计算当前帧的宏块地址。而对于非内部宏块，不光要计算当前帧的宏块地址，还要计算预测帧的宏块地址。如果是双向预测要计算两个预测帧的宏块地址。

在 MPEG 图像压缩系统中，输入图像数据是按照宏块、宏块条、图像的分层结构组织的，当前帧图像被重构并按一定的顺序以宏块条的形式写入帧存。在一个 16*16 宏块中，有 6 个 8×8 块。令一个宏块的左上角是这个块的起始地址，那么在一个宏块中 6 个块的起始地址序列是一个固定的地址。在一个帧内的宏块条的起始地址序列也是一个固定的地址。

以 352×288 分辨率下的图像为例，讨论如何计算 MPEG-2 视频码流的参考宏块的起始地址[40]。首先，从一帧图像的第一行开始，每连续的 16 行象素作为一个象素块，称为行块，图像 18 个行块。其次，每一个行块的水平方向上连续 16 列象素组成列块，所以图像可划分为 22 个列块。在码流中有 slice_vertical_position 这个参数，这个参数给出了以宏块数为单位的第一个宏块在宏块层的垂直位置，即宏块的行块号。另外码流中还有 macroblock_address_increment 这个参数，它指出了当前宏块(macroblock_address)与前一宏块(previous_macroblock_address)的地址差值。在一个组块的起始处，previous_macroblock_addr 复位成（行块号×22－1）。所以预测宏块的起始行列地址为（行块数×16，列块数×16），在加上运动向量（vector_h，vector_v），就得到参考宏块的行列坐标（行块数×16＋vector_h，列块数×16＋vector_v）。

4.8.2 象素缓冲器

它是帧存储器和象素预测单元之间的接口电路，因为运动补偿电路需要频繁访问帧存储器，但存储器的读写效率有限，所以其操作成为限制运动补偿的瓶颈，所以充分利用硬件电路的优势，设计象素缓冲器这个单元，可以加快从帧存储器中读取象素和预测象素产生单元产生数据的效率。

4.8.3 预测单元

一个给定的象素通过读取参考帧中有运动向量的对应象素来进行预测的。在预测过程中，参考宏块可能经过半象素操作。前向预测、后向预测或双向预测产生预测宏块，再与 IDCT 后的残差数据相结合，最终得到重建象素。在第二章已经提到，MPEG-2 标准中，所有运动矢量都是半象素的，所以前向预测和后向预测都要进行半象素操作，如果同时进行前后向预测还要平均滤波。为了减轻运算的负担，此单元采用了 4 个象素缓冲器并行处理数据，这样大大节省了象素缓冲器的大小和预测时间。

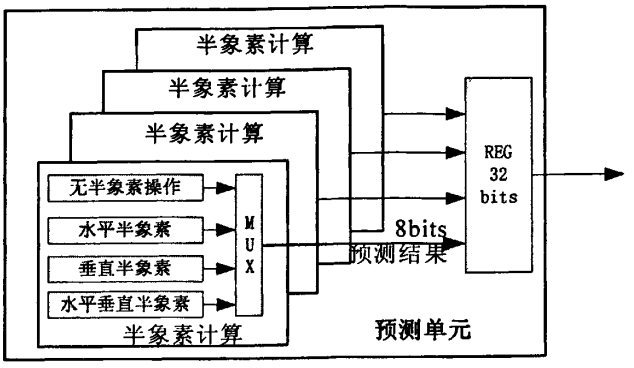

图 4-18 预测单元结构图

当以上的操作完成，可以说运动补偿的计算已经完成。剩下的工作就是图像的重建，这是 MPEG-2 解码的最后一步，就是把运动补偿的预测结果与 IDCT 的输出的数据相加，并对结果进行取模运算，最终形成解码样本。

MUX 单元：判断重建图像类别，如果解码图像为 B 帧，则把数据直接送入显存显示，因为 B 图不能作参考帧。如果为 I、P 图，则写入帧存。

帧存：存储 I 帧和 P 帧 YUV 数据。大小为 32bit×256k 的 SRAM。

4.9 后处理模块

在 MPEG-2 视频标准中，图像信息采用亮度信息和两个色差信息。因为人眼对色度的分辨率较之亮度的分辨率低，通常降低色度信号的采样频率[41]，来实现数据的压缩，YcbCr 有多种采样格式，其中 4：2：0 格式的视频信号应用最为广泛。在这种采样方式下，色度信号在水平方向和垂直方向的采样率都只有亮度信号采样率的一半。这样就可以大大节省解码系统的存储空间和带宽，并且简化了计算过程。当整个系统完成解码运算后，必须把数据转换为 RGB 色彩空间的数据，之后才能通过 VGA 口显示出去。

从 YcbCr 到 RGB 的色度空间转换用到的计算公式如下：

R=1.164(Y-16)+1.596(Cr-128)

G=1.164(Y-16)-0.813(Cr-128)-0.392(Cb-128)

R=1.164(Y-16)+2.017(Cb-128)

在硬件设计时，所有的计算都采用定点整数的方法，所以在程序中都需要把上述这些公式的小数全部扩 256 倍，相当于左移 8 位。

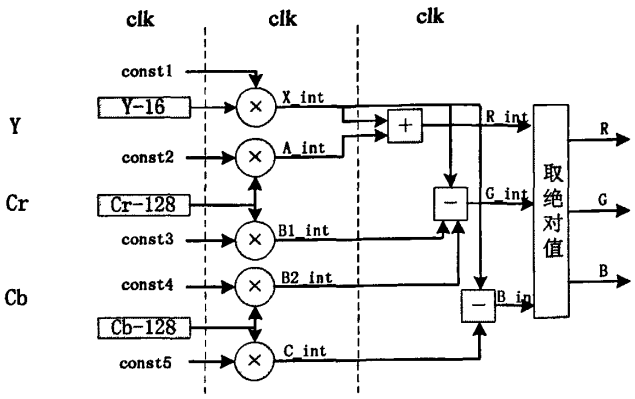

图 4-19 颜色空间转换电路结构图

从图上可以看出，完成一个象素的这样操作需 3 个时钟周期，输出为 8bit 的 R、G、B 值。

4.10 本章小结

本章详细介绍了 MEPG-2 视频解码器的硬件设计方法，描述了主要模块的硬件设计实现原理和过程，并给出部分模块的仿真波形并作出解释。下一章节就可以直接进行系统的原型验证和逻辑综合了。

第五章　解码器的验证和综合

5.1 基于 FPGA 的验证

5.1.1 验证的意义和定义

目前数字专用芯片的集成度和主频速度呈摩尔定律以每两年翻一倍的高速发展，设计的复杂性导致了设计周期的拉长。尽管 EDA 行业试图通过引入工具来帮助提高设计效率，缩短设计时间，但功能复杂度的增长速度更快，流片后芯片的功能性错误率也在不断上升。设计错误将会带来研发周期的延长，且芯片的研发成本也不断提高，给市场预测带来了更多的风险和不确定性。

在这样的前提下，怎样减少设计中的错误，这成为芯片设计中一个越来越突出的问题。在设计流程中引入验证，用于检测设计的正确与否，也就成了必然。如何对设计中的复杂功能进行验证已经成为缩短产品时间所需要面对的挑战。根据统计，数字系统设计的测试验证所至少占据了 70%的设计工作量[46]，而且验证能力远远落后于设计能力，因此减少验证所花费的成本，提高仿真验证效率成为当前数字电路设计的关键。

就仿真验证而言，主要可以分为两大类方法:软件仿真、基于硬件加速器或 FPGA 原型的硬件仿真。对于超大规模的硬件设计，其设计功能的提升导致其需要测试的向量数目成指数增长,这使得验证工作量随设计复杂度增长而迅速增加、验证周期大幅延长，显然不是设计者所希望看到的。硬件加速/仿真器则有比较好的性能，它可以帮助设计人员在设计早期建立芯片的硬件模型、可以帮助设计人员在设计早期调试芯片软件、设计和可以很方便的使用的内置逻辑分析仪对电路进行调试，但是它只适合于特定的仿真，不仅昂贵，而且很难满足 ASIC 的所有性能要求。对比以上两种方法，越来越多的公司和设计人员采用 FPGA 进行功能验证。原因如下：

1. FPGA 原型验证可以使我们找出其他验证方法不易发现的错误；

2. FPGA 原型验证的速度接近于芯片实际速度，可以使我们尽早地来测试应用软件；

3. 随着大容量高性能 FPGA 的出现，建立一个高性价比的 FPGA 原型验证系统要比其他的方法更加便宜和快速。

5.1.2 FPGA 的选择

根据设计的需要和现有的条件,在本次设计中采用了 Altera 公司的 stratixII 系列中的 EP2S180 开发板[42]作为载体。StratixII 系列发布于 2004 年,采用 TSMC 90nm 制造工艺，9 层金属，1.2V 内核电压。

它的内部主要特性有：内嵌 RAM 块、DSP 块、锁相环（PLL）和外部的存

储器接口、专门用于音视频的转换接口等。

表 5-1 stratixII 系列功能表

功能	EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180
自适应逻辑模块（ALM）	6,240	13,552	24,176	36,384	53,016	71,760
等效逻辑单元（LE）	15,600	33,880	60,440	90,960	132,540	179,400
M512 RAM 块（512 bits）	104	202	329	488	699	930
M4K RAM 块（4 Kbits）	78	144	255	408	609	768
M-RAM 块（512 K）	0	1	2	4	6	9
总共 RAM bits	419,328	1,369,728	2,544,192	4,520,448	6,747,840	9,383,040
DSP块（每个DSP包含4个18x18乘法器）	12	16	36	48	63	96
锁相环（PLL）	6	6	12	12	12	12
最大可用I/O管脚	358	542	702	888	1,110	1,158

开发板采用的一些具体技术有：

1. TriMatrix 存储器及外部存储器接口

它是 Stratix II 器件具有革命性的创新设计。TriMatrix 存储器由三种不同大小的 RAM 块组成：512bit 的 M512 块、4Kbit 的 M4K 块以及 512Kbit 的 M-RAM 块。基于这三种块的 RAM 单元总容量达到 9Mbits，所以 TriMatrix 存储器提供了多种结构来实现各种各样的存储器函数。而通过 Stratix II 的专门外部存储器接口可访问外部多种高速存储器，包括：DDR、DDR2、SDRAM、SRAM 等。

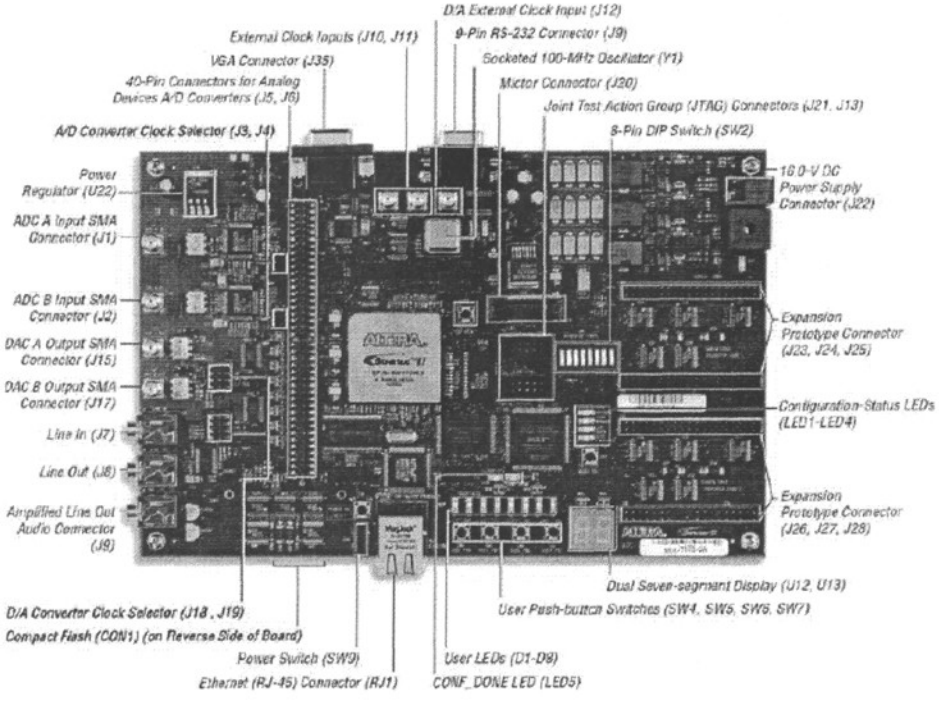

图 5-1 EP2S180 开发板元件和接口图

2. 数字信号处理器（DSP）模块

因为 DSP 具有很强的运算能力，而 Stratix II 器件最多带有 96 个 DSP 块。每个 DSP 块可配置成 8 个 9×9 或 4 个 18×18 或者 2 个 36×36bit 的乘法器。

此外，每个 DSP 块有 4 种操作模式：简单乘法、乘加、有两个乘法器的加法器和有四个乘法器的加法器。这些模式的组合使得 DSP 块能够很方便的实现 FFT 等复杂运算。

上述技术的使用，使得 StratixII 系列器件既满足现今高级系统的高性能要求，又避免了使用昂贵的 ASIC 进行开发[42]，在无线通信、图像处理、高速数字信号处理和军事雷达等领域都有广泛的应用前景。

5.1.3 VGA 标准和控制电路的设计

为了更直观的对 MPEG-2 视频解码器进行验证，可以利用开发板上的 VGA 接口，进行数据的 D/A 转换，将视频数据输出到显示器上进行观察。

VGA（Video Graphic Array）接口是与显示器进行通信的唯一接口。它采用非对称分布的 15 针连接方式。其工作原理是：将符合一定格式下扫描时序要求的视频数据流经过模拟调制转换成模拟高频信号，然后再输出到显示设备成像。VGA 显示器采用逐行扫描的方式，电子束按照固定路径扫描整个屏幕。扫描从屏幕的左上方开始，从左到右，从上到下，逐行扫描。其过程为：电子束从屏幕左上角开始向右扫，当到达屏幕的右边缘时，电子束关闭（水平消隐），并快速返回屏幕左边缘（水平回扫），然后在下一条扫描线上开始新的一次水平扫描。一旦所有的水平扫描全部完成，电子束在屏幕的右下角结束并关闭（垂直消隐）。

通过 FPGA 器件控制 RGB 信号、行同步信号、场同步等信号，并参照有关标准，最后可以实现对 VGA 显示器的控制[43]，显示频率为 38.25MHz。实现的电路图如 5-2 所示。

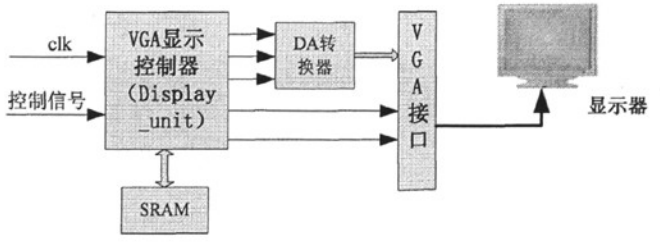

图 5-2　VGA 显示控制电路图

在显示单元的设计里，存在行、场同步信号的计数器，在图像有效区间内按照图片的大小规定行和列的起始坐标和终止坐标。当扫描信号到达显示图片的区间时，图像的地址计数器开始计数，并从帧存或显存里相应的位置读出数据送入显示器进行显示。

5.1.4 验证策略和结果

在进行 FPGA 验证之前，必须对设计进行前仿真，这样可以找出设计中的错误并进行修改。具体过程如下：我们对整个解码系统搭建一个测试平台，把一段输入视频码流预存入 RAM 中，启动整个解码模块，当解完一帧后，从 RAM 中导出数据，还原成图片后可以检查图片中是否存在问题，如有错误，则返回

仿真波形图找出问题出现的原因以修改设计，如此往返操作可以比较容易的调试设计的程序。可以这样调试的原因是因为图像中的问题都是以宏块为单位出现的，而人眼对宏块等级的图像已经很敏感了，所以满足调试的需求。图 5-3 所示为仿真所得到的解码图像。

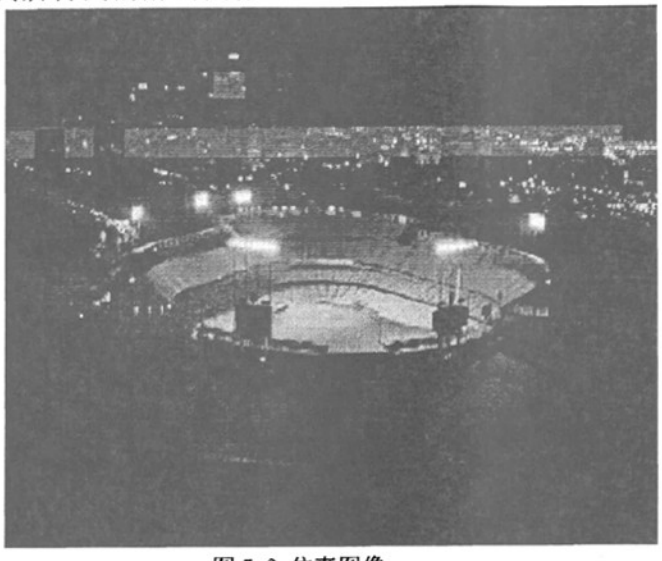

图 5-3 仿真图像

解码的图像规格为 352*288，每个宏块有 16 个象素，所以图像长由 22 个宏块、宽由 16 个宏块组成。由仿真图像可知，解码得到的图像数据存在明显错误，错误的宏块包括第 3 行第 8 列的宏块、第 5 行的大部分宏块，在确定图像的错误地点后，查看波形图，找出宏块相应位置的波形数据，具体检查 6 个块数据的解码、计算的结果，找出和参考软件的结果数据不一致的地方，并确定错误的产生出自哪个步骤，这样就可以返回程序找出产生错误的源代码，并作正确的修改，如此循环测试之后，就可保证图像解码的正确了。

我们用 EDA 工具 modelsim 在对图像解码时，获得一帧图像大概需 4~6 分钟，若要对大流量的数据进行测试，只修改一个错误就将消耗大量的时间，这是得不偿失的，所以对大流量的图像验证就采用 FPGA 验证。

此时整个解码器的 FPGA 验证方案为，首先把编码的视频码流通过网口下载到开发板上的 SDRAM 中，然后解码器读入数据进行解码操作，并将解码后的数据按顺序存入帧存 SRAM 中（存 I 帧和 P 帧）或显存中（存 B 帧图像），最后通过 VGA 接口输出到显示器，对运动图像进行显示。我们以电影《变形金刚》为解码原型，解码后的运动图像如图 5-4 所示。

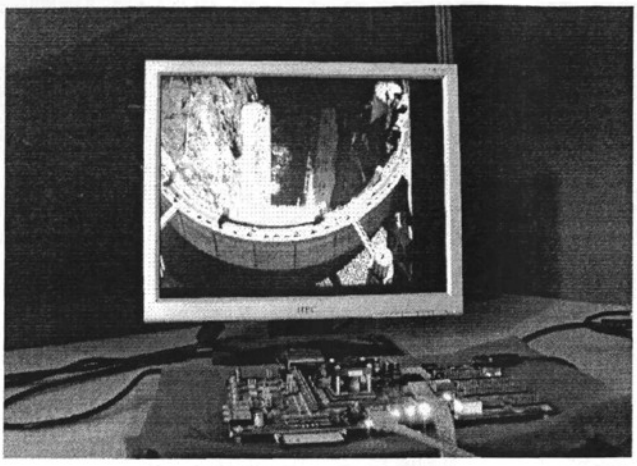

图 5-4 验证结果图(a)

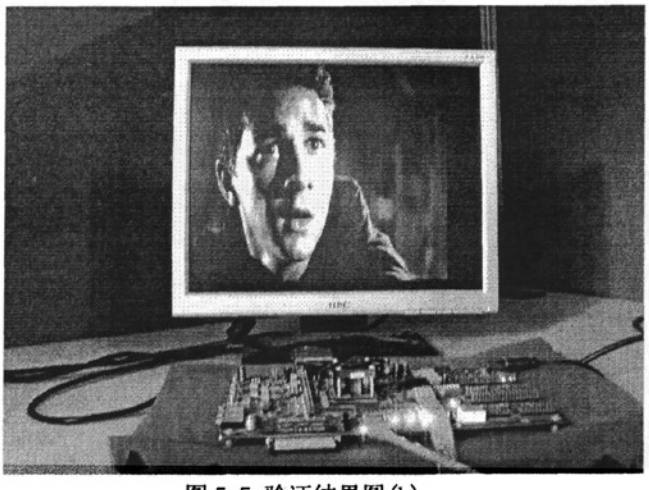

图 5-5 验证结果图(b)

表 5-2 FPGA 消耗资源参数报告

消耗资源参数	数据
Logic utilization	7 %
Combinational ALUTs	8,313 / 143,520 (6 %)
Dedicated logic registers	5,280 / 143,520 (4%)
Total registers	5280
Total pins	96 / 743 (13 %)
Total block memory bits	5,284,833
DSP block 9-bit elements	97
Total PLLs	1

整个解码器所消耗的 FPGA 资源占总资源的 7%，其中组合逻辑占 8362 个逻辑单元，寄存器占 4790 个，另外还消耗了 5284833bit（占总存储的 56%）的存储资源和 97 个 DSP 模块。整个解码模块所能达到的频率约为 78MHz。

5.2 解码器的逻辑综合

综合，是从逻辑设计到电路实现的第一步[44]。在 RTL 设计中，就要充分考虑描述的可综合性问题。可综合性指的是一个电路描述的综合收敛性，换句话说，一个电路描述在很大程度上可以由 EDA 软件自动生成合情合理的电路实现。系统级描述基本上不具备综合性，因为它们过于抽象，导致了太多的综合随意性，因此纯行为描述的可综合性很差。RTL 级描述的综合性就很好了，而且越往底层综合性就越好。

从上述可得，可综合性与电路描述的抽象程度密切相关。在电路设计实践中，设计人员总是从抽象的系统级描述或行为描述开始，用仿真工具验证顶层设计的正确性，这个阶段定义顶层各个功能块的外在特性，但还没有功能块的内容，因此不能综合。设计工程师从系统工程师手中接收到顶层功能块的外特性，并开始着手设计它的 RTL 内容，这时就要考虑描述的可综合性问题，因为综合的对象是 RTL 级描述的程序。

就现有的 EDA 工具而言，逻辑综合就是将 RTL 级的描述转换到门级网表的过程。通过逻辑综合，设计者可以在布局布线之前优先考虑约束问题，尽早发现设计的违规情况，而不将其带入物理设计。设计者可以通过对设计施加不同的约束条件，得到时序和面积充分优化的综合结果，对于数字电路来说就是在电路的面积和功耗、面积和时序性能上的折衷。

综合过程主要包含三个阶段：转换（translation）、优化（optimization）和映射（mapping）。转换阶段综合工具将高层语言描述的电路用门级的逻辑来实现，对于业界普遍使用的综合工具 DC 来说，就是用工艺库中的门级单元来组成 HDL 语言描述的电路，从而构成初始的未优化的电路。优化和映射是综合工具对已有的初始电路进行分析，去掉电路中的冗余单元，并对不满足限制条件的路径进行优化，然后将优化的电路映射到单元库上。综合过程如图 5-6 所示。

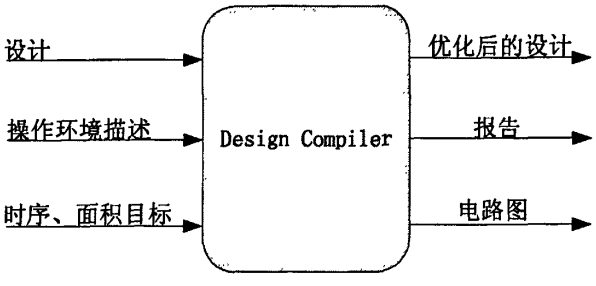

图 5-6 综合流程

5.2.1 定义设计环境

在对设计进行优化前，必须模拟出设计与其工作的环境。通过制定操作条件、线性负载模型和系统接口特征来定义环境。

本文设计的 MPEG-2 解码器的环境约束如下图 5-7 所示。

```
set_operating_conditions WORST
set_wire_load_model -name reference_area_25000000
set_wire_load_mode top
set_max_area 25000000
set_drive 0 [list clk rst_n]
set_input_transition 2.0 [all_inputs]
set_fanout_load 3 [all_outputs]
```

图 5-7 环境设计脚本

1．操作条件包括电压、温度和方法的变更，对于绝大多数工艺，上述操作条件的变化会对电路有相当重要的影响。综合库一般包括最差（WORST）、典型（Typical）、最好（BEST）三种环境，在综合阶段使用最差环境来优化设计，因为此环境下电路要求最大的建立时间，如果此时的设计满足要求，则其他环境下建立时间要求必然满足。使用最好环境来清除保持时间的违规，因为此环境下两触发器之间的逻辑延迟最小，如果此时的设计满足要求，则其他环境下的保持时间要求必然满足。

2．Set_wire_load_model（线形负载模型）的定义估计了线长和扇出对于电阻、电容和线的面积的影响程度。DC 利用这些物理值来计算线延迟和电路速度。在综合阶段，线形负载模型应相当悲观，这样可以给布局布线留下一定的冗余时序，为物理设计提供较多的时序空间。对于穿越层次界限的线，DC 工具还支持其线形负载模型三种工作模式：Top（顶部）模式表示层次设计中的线延迟按照最顶层模块的线形负载模型来计算，忽略所有子模块的线形负载模型；Enclosed（依附）模式表示子设计的线延迟按照直接调用它们的子系统（设计）来计算；segmented（分割）模式表示穿越层次界限的线的线形负载模型计算根据每一层次的具体定义来进行计算。

3．系统接口建模，DC 可以通过对输入端口定义驱动特性、对输入和输出端口定义负载、对输出端口定义扇出负载来来模拟设计和外部系统的接口，比如 Set_fanout_load 用来指定输出端口预期的扇出负载值，DC 试图确保输出端口的扇出负载的总和加上与输出端口驱动器连接的单元的扇出负载要小于库、库单元和设计的最大扇出限制。

逻辑综合除了对设计进行工艺的映射外，更重要的是对设计的时序、面积以及性能进行合理的优化，使得设计在时序、面积和性能达到最优化的折中。在对设计进行优化时，DC 支持两种类型的约束：设计规则约束（Design rule constaints）和最优化约束（Optimization constraints）。

5.2.2 设计规则约束

设计规则约束是固有的，在工艺库里定义，DC 给设计对象赋予属性来表示这些设计规则约束，就是为了保证设计的功能正确性。最常用的设计规则约束为：

转换时间（Transition time）约束：线的转换时间约束是对它的驱动管脚改变逻辑值的时序要求。命令 set_max_transition 用来改变工艺库里指定的最大转换时间约束，这条命令给设计中所有线或与确定端口相连的线设置了最大转换时间。在最优化的过程中，dc 试图使每一条线的转换时间都小于 max_transition 属性值，如图 5-8 所示。

```
set MAX_TRANSITION  3
set_max_transition $MAX_TRANSITION [all_inputs]
set MAX_CAPACITANCE  3
set_max_capacitance $MAX_CAPACITANCE [all_inputs]
```

5-8 设计规则约束脚本

扇出负载（Fanout load）：线的最大扇出负载是指这条线所能驱动的最大数目的负载。在最优化时，DC 试图满足每一个驱动管脚的扇出负载限制，如果一个管脚违反了此限制，DC 会尽力改正这个问题。Set_max_fanout 命令对设计或输入管脚设置比工艺库里指定的更为保守的扇出约束，这样 DC 会尽力满足更小的扇出限制。

电容（capacitance）：它的属性有最大电容和最小电容。通过 Set_max_capacitance 命令给一个单元的输入端口或管脚来设置最大电容约束。在编译的过程中，DC 要确保最大的电容没有违规，即驱动单元输出端口上的电容值大于等于被驱动单元端口上的电容值之和。同样也可以用 set_min_capacitance 命令对输入端口或管脚来设置最小电容约束。

DC 工具还支持最优化约束，这由设计人员定义，用来描述设计指标，在整个脚本工作期间应用于当前设计。最优化约束最常用的约束包括时序约束和面积约束。

5.2.3 设置时序约束

时序约束指定了设计所要求的性能。其约束主要包括以下几个方面：定义时钟的周期和波形：利用 create_clock 来定义时钟的周期和波形，这是非常重要的约束，是其他时序路径上定时约束的基础，其约束条件如图 5-9 所示。

```
set PERIOD 15.0
set HALF_P [expr $PERIOD / 2.0]
set UNCERTAINTY  0.5
set DELAY 1

create_clock   -period $PERIOD  -waveform [list 0.0 $HALF_P] {clk}

set PORT_In_To_CK  [expr $PERIOD * 0.10]
set_input_delay  $PORT_In_To_CK -clock  clk  [all_inputs]
set PORT_CK_TO_OUT  [expr $PERIOD * 0.10]
set_output_delay  $PORT_CK_TO_OUT -clock  clk  [all_outputs]

set_clock_transition 1.2 clk
.......................
```

图 5-9 时序约束脚本

因为它对设计中所有寄存器到寄存器的路径加以延时。此外，考虑到由于布局导致的时钟网络的变化，利用 set_clock_uncertainty 命令的-setup 或-hold 选项增加少许的时序余量。指定 I/O 时序要求：在绝大多数情况下，信号的输入/输出的时间是交错的。可以用以下命令来达到以上目的。set_input_delay 定义输入端口的到达时间，相对于系统时钟和其他输入端口的输入延迟约束；set_output_delay 定义要求的输出到达时间，这是相对于系统时钟的输出延迟约束；对于不被时钟周期限制的组合延迟，用 set_max_delay 和 set_min_delay 命令定义指定路径的最小和最大延迟。此外还有指定错误路径、设置多周期路径等约束设置。

5.2.4 设置面积约束

用 set_max_area 命令对当前设计设置一个 max_area 属性，它是由每一个元件和线的面积组成，用工艺库里面积的单位来指定这个面积。

时序约束和面积约束通常称为最优化约束，当两者同时定义时，Design Compiler 将首先满足时序约束。但最优化约束和设计规则约束相比时，Design Compiler 虽然试图同时满足两约束，但设计规则约束必须首先被满足。

5.2.5 综合报告

报告中最基本的是面积报告和时序报告，对这些报告的分析能够使设计者了解设计中的不足而对设计进行相应的修改。

```
****************************************
Report : area
Design : top
Version: 2002.05
Date   : Tue Sep 16 23:38:48 2008
****************************************

Library(s) Used:

    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/ram_in_xl.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/fifo_out_xl.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/ram_fifo_xl.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/quantiser_ram.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/dis_ram.db)
    smic18_ss
(File:/project/library/smic18/FEView_STDIO/Version2.0/STD/Synopsys/smic18_ss.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/idct_ram.db)
    USERLIB (File: /export/home1/2006/mal/maliang/dc/syn_all/db/dct_ram.db)

Number of ports:              91
Number of nets:             1815
Number of cells:             133
Number of references:         27

Combinational area:       666949.312500
Noncombinational area:    12033860.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:          12700907.000000
```

图 5-10 综合的面积报告

由上面报告可知，用 smic 0.18 工艺库综合出来的总面积为 12700907um^2。

综合的时序报告如图 5-11 所示：

```
****************************************
Design : top
Version: 2002.05
Date   : Wed Sep 17 09:58:46 2008
****************************************

  Point                                                   Incr       Path
  ------------------------------------------------------------------------
  clock clk (rise edge)                                   0.00       0.00
  clock network delay (ideal)                             1.00       1.00
  vld/motion_type_reg[0]/CK (FFSJKRHD1X)                  0.00 #     1.00 r
  vld/motion_type_reg[0]/Q (FFSJKRHD1X)                   1.25       2.25 r
  vld/motion_type[0] (vld)                                0.00       2.25 r
  mc/motion_type[0] (mc)                                  0.00       2.25 r
  mc/U862/Z (NAND2B1HD4X)                                 0.35       2.60 f
  mc/U910/Z (INVCLKHD80X)                                 0.66       3.27 r
  mc/addr_computer2_bwd/motion_field (addr_computer_0)
                                                          0.00       3.27 r
  mc/addr_computer2_bwd/U332/Z (MUX2HD1X)                 0.62       3.89 f
  ------------------------------------------------------------------------
  mc/addr_computer2_bwd/add_145/SUM[13] (addr_computer_0_DW01_inc_14_2)
                                                          0.00       7.22 r
  ------------------------------------------------------------------------
  mc/addr_computer2_bwd/U529/Z (OAI31HD2X)                0.20       7.42 f
  ------------------------------------------------------------------------
  mc/addr_computer2_bwd/mult_308/PRODUCT[19] (addr_computer_0_DW02_mult_10_14_0)
                                                          0.00      15.00 r
  mc/addr_computer2_bwd/U192/Z (AND2HD1X)                 0.32      15.32 r
  mc/addr_computer2_bwd/tmp1_reg[19]/TI (FFSJKRHD1X)      0.00      15.32 r
  data arrival time                                                 15.32

  clock clk (rise edge)                                  15.00      15.00
  clock network delay (ideal)                             1.00      16.00
  clock uncertainty                                      -0.50      15.50
  mc/addr_computer2_bwd/tmp1_reg[19]/CK (FFSJKRHD1X)      0.00      15.50 r
  library setup time                                     -0.18      15.32
  data required time                                                15.32
  ------------------------------------------------------------------------
  data required time                                                15.32
  data arrival time                                                -15.32
  ------------------------------------------------------------------------
  slack (MET)                                                       0.00
```

图 5-11 综合的时序报告

报告中包含了路径的起点、中间点和终点，路径上的每个点使得延迟慢慢积累，到达终点的时间即为数据到达时间。图 5-11 所示的是电路中时序最差的一条路径，也是电路的关键路径，此信号由 VLD 模块产生，送入运动补偿（MC）模块，经过一定的组合逻辑，送入子模块 addr_computer1_fwd，因为加法器和乘法器的性能的限制，使得整个电路的时钟频率为 66Mhz。

5.3 本章小结

本章在 RTL 级硬件设计的基础上，完成了解码器的 FPGA 原型验证与综合，并给出它们的结果。

第六章　总结与展望

6.1 总结

本文主要研究的是 MPEG-2 视频解码的 FPGA 设计，并以此为基础，详细描述了 MPEG-2 视频解压缩的算法原理、软件实现、系统架构、电路设计及优化等方面的内容，最后并对整个设计进行了仿真测试和 FPGA 验证。完成的主要工作包括以下几个方面：

1. 完成解码系统的体系结构的设计，采用了自顶而下的设计方法，实现系统的功能单元的划分；根据其视频解码的特点，确定解码器集中控制的方式；实现帧内数据和帧间数据的并行解码。
2. 根据解码器的设计，比特流格式器模块设计提出了特有的解码方式；在可变长模块中的变长数据解码采用组合逻辑外加查找表的方式实现，大大减少了变长数据解码的时间；IQ、IDCT 模块采用流水的设计方法，减少数据计算的时间；运动补偿模块，针对模块数据运算量大和访问帧存储器频繁的特点，采用四个插值单元同时处理，增加像素缓冲器，充分利用并行性结构等方法来加快运动补偿速度。
3. 根据视频解码的参考软件，通过解码系统的仿真结果和软件结果的比较来验证模块的功能正确性。最后用 FPGA 开发板实现了解码系统的原型芯片验证。

整篇文章对 MPEG-2 视频解码技术展开了较深入的研究，并结合了其他的一些设计方法，设计出具有一定特点的 MPEG-2 视频解码器的 FPGA 原型芯片，取得良好的解码效果。

6.2 展望

本文设计的视频解码器完成了实现了预期目标，取得了不错的解码效果。但是在本文的基础上，还能在以下几方面进行优化：

1. IDCT 模等模块尽管内部计算采用的是流水计算的方法，但还可以采用更好的算法实现。
2. 一些功能单元的计算用流水计算实现，但模块间的计算没有采用，在以后的优化设计中可以实现 Huffman 模块、IQ 模块和 IDCT 模块的流水计算。
3. 本文研究的解码系统中采用的是集中控制的模式，以后可以进行控制策略的改良。

参考文献

[1] 尹显东、李在铭、姚军等，图像压缩标准研究的发展与前景[J] 信息与电子工程，2003

[2] 余松煜、张 文军、孙军，现代图像信息压缩技术，科学出版社，1998

[3] 罗森林、潘丽敏，从 MPEG-1 到 MPEG-21，世界广播电视，2001

[4] 李斌、李伦，MPEG-1～MPEG-21 的分析与评价，电视技术，2001

[5] 余兆明、李晓飞、陈来春，MPEG 标准及其应用，北京邮电大学出版社，2002

[6] 钟玉琢、王琪、赵黎、杨小勤，MPEG-2 运动图像压缩编码国际标准及 MPEG 的新进展 清华大学出版社，2002

[7] Iain E.G.Richardson（美），H.264 和 MPEG-4 视频压缩(新一代多媒体的视频编码技术)，2004

[8] Sundarajan.Sriram， Ching-Yu.Hung， MPEG-2 video decoding on the TMS320C6X DSP architecture，IEEE，1998

[9] David A.Bader，Sulabh.Patel，High Performance MPEG-2 Software Decoder on the Cell Broadband Engine，IEEE，2009

[10] 俞绍安，基于 MPEG-2 的软件解码器，中国科技大学 硕士学位论文，2002

[11] 富士通微电子推出标清多路解码器芯片

[12] ISO IEC MPEG-2 13818-5 CHS

[13] 王清，MPEG-2 视频解码器的系统级实现，同济大学 硕士学位论文，2006

[14] C.T.Hish，S.P.Kim，A Concurrent Memory-Efficient VLC Decoder for MPEG Application，IEEE，1996

[15] H.C.Chen，Y.L.Wang，Y.F.Lan，A Memory-Efficient and Fast Huffman Decoding Algorithim，Information Processing Letters，1999

[16] K.L.Chung，J.G.Wu，Level-Compressed Huffman Decoding，IEEE，1999

[17] 邱临海，余胜生，周敬利，快速 Huffman 解码算法及其实现，计算机工程与应用，1999

[18] R.Hashemian，Design and Hardware Implementation of a Memory Efficient Huffman，IEEE，1994

[19] Hashemian，Memory Efficient and high-speed Search Huffman Coding，IEEE，1995

[20] 朱翠涛，陈少平，陈亚光，并行 Huffman 解码算法的设计与实现，计算机测量与控制，2002

[21] R.Hashemian，Condensed Huffman Coding，A New Efficinet Decoding Technique，IEEE，2002

[22] R.Hashemian，Direct Huffman Coding and Decoding Using The Table of

[22] R.Hashemian，Direct Huffman Coding and Decoding Using The Table of Code-Length，IEEE，2003

[23] W.H.Chen、C.R.Smith、S.C.Fralick，A Fast Computational Algorithm for the Discrete Cosine Transform，IEEE，1977

[24] ZhongDe Wang，Fast Algorithm for the Discrete W-Transform and for the Discrete Fourier Transform，IEEE，1984

[25] Byeong Gi Lee，A New Algorithm to Compute the discrete Cosine Transform，IEEE，1984

[26] N.Suehiro，M.Atori，Fast Algorithm for the DFT and other Sinusoidal Transforms，IEEE Transactions on Acoustics，Speech，and Signal Processing，1986

[27] H.S.Hou，A Fast Recursive Algorithm for Computing the Discrete Cosine Transform，IEEE Transactions on Acoustics，Speech，and Signal Processing，1987

[28] 王蕙、虞露、毛讯等，MPEG2 专用视频解码 VLSI 中的控制策略，电路与系统学报，2002

[29] Nam Ling，etl An Efficient Controller Scheme for MPEG-2 Video Decoder，IEEE，1998

[30] Masaki，toyokura et al， A single-chip Real-Time MPEG-2 Video Decoder，IEEE，conf consumer Electro，1994;

[31] 惠新标，郑志航，叶楠 MPEG-2 视频解码控制的 VLSI 设计 通信与电视 1998

[32] 徐晶,李炜,冯宇红 基于 MPEG-2 视频基本流解码的容错方法研究 计算机应用研究 2003

[33] 杜晓刚、秦东、何寅等，适用于 MPEG-2 视频解码的 VLD 设计，微电子学，1999

[34] 李志俊、蔡敏、郑学仁，MPEG-2 视频解码的可变字长解码器的设计，华南理工大学学报(自然科学版)，2001

[35] Latha Pillai，Quantization/Xapp615(Xilinx)，2003

[36] 王纲，8×8 整型 DCT/IDCT 变换算法研究，实验科学与技术，2006

[37] 饶海潮、郭立、黄征，IDCT IP 核的 VLSI 结构，微电子学与计算机，2004

[38] Latha Pillai，Video Decompression Using IDCT/Xapp611(Xilinx)，2007

[39] 惠新标，郑志航，叶楠，MPEG-2 运动补偿的 VLSI 设计，上海交通大学学报，2000

[40] 舒海军，MPEG-2 解码器中运动补偿的 VLSI 实现，上海大学 硕士学位论文，2004

[41] 张海波、李方伟、刘开健，颜色空间转换电路的 FPGA 设计与实现，重庆职业技术学院院报，2008
[42] StratixII EP2S180 DSP Development Board Reference Manual，2005
[42] 王诚、吴继华、范丽珍，Altera FPGA/CPLD 设计，人民邮电出版社，2005
[43] 扬杰、穆伟斌、沈焕泉，基于 FPGA 的 VGA 控制器设计与实现，齐齐哈尔大学学报，2008
[44] Bhatnagar.H，高级芯片综合，清华大学出版社，2007
[45] 胡力佳、马琪、徐向阳，数字集成电路设计中的硬件加速验证技术，现代电子技术，2007