摘 要

现代社会急速向信息社会发展，而多媒体技术是信息化中主要的技术环节之一。目前，多媒体技术的发展，多媒体技术已渗透到各个学科领域和国民经济的各个方面。伴随着多媒体处理技术的标准化，有很多函数模块都是几乎一样的，这就造成了在多媒体处理系统开发过程中存在很多的重复工作，耗费大量人力物力，而且开发时间也很长。为了降低多媒体系统开发的难度，本文提出了以这些通用模块为主体的多媒体处理库，供多媒体系统开发者调用。

目前，多媒体系统的实现主要有 3 类方法：一类是基于 PC 机；二是使用专门的多媒体 ASIC 芯片；还有可编程通用 DSP。数字多媒体处理器是一种专门针对多媒体处理的 DSP，为了适合多媒体数据处理，其处理器结构经过了特别优化设计，而且提供了丰富的多媒体处理指令集。DM642 是 TI 公司新一代高性能多媒体处理器，它是专门用于多媒体系统应用的高性能 DSP，运算能力达到 4800MIPS，而且具有丰富的外围接口，是目前多媒体系统实现的理想平台之一。

本文首先介绍了多媒体库框架及实现的函数类，对主要函数进行了分析，特别是对多媒体函数进行分层封装，用户可以根据对多媒体的了解程度和需要来调用相应的函数。然后详细介绍了多媒体库优化的物理基础——DM642。重点介绍了体系结构，指令执行流程，指令集。在简单介绍了视频处理库的优化软件平台 CCS 之后，从 MPEG4 以及 H.264 编码器结构调整，存储器的优化，代码优化 3 个方面对编解码库优化方法进行了重点介绍和分析，并给出相应的实验优化结果。最后对本文的工作进行了总结，并提出了今后的工作方向。

关键词 多媒体处理库，DM642，优化，编解码

Abstract

The modern society is rapidly developing to information society, while multimedia is one of the most important technologies during the development. Multimedia technology has merged into many aspects of the national economy. With the standardization of multimedia processing techno-Logy, some module, such as the SAD, exist in always every multimedia system. As the result, we conclude a multimedia processing library based on DSP, which facilitate the multimedia system development based on DSP.

Cuurently, the implementation of the multimedia system is as follows: (1) multimedia system based on the PC. (2) using the ASIC chip for multimedia system. (3) using the DSP especially for the digital multimedia. The hardware structure of DSP processor core is especially designed for multimedia data processing. Not only plentiful multimedia data processing instructions are available, but also it is more flexible for implementation of multimedia syatem processing. The implentment using DSP can shorten the time to market and offer convenience for upgrading and reprogramming. DM642 is the new generation data signal processor especially for multimedia processing, which offer s industry-leading performance. At the 600 MHz, DM642 offers the processing capability of the 4800 MIPS. So, DM642 is an ideal platform for multimedia applications.

First, the multimedia processing library is proposed. The multimedia processing library is encapsulated in several level functions. The lower function can constructed the higher function. Secondly, DM642, which is the physical basement for the optimization of the library, is anlaysed. And the follows is the emphases of the paper. After introducing CCS, which is software platform for the optimization, the motheds, which are used to optimize the multimedia processing library are presented. The methods cover three aspects: the adjustment of CODEC framework; the setting of memory allocation; the optimization of the codes. In the end, there is about conclusion and tasks in the future..

Key words: MML, DM642, optimization, CODEC

第一章 音视频编码技术概述

1.1 引言

伴随着微电子技术的不断发展，各种多媒体的处理技术也有了长足的进步，从音视频压缩，语音识别，到图像识别无不成为目前的研究热点。经过十几年乃至几十年的研究，很多的多媒体技术已经比较成熟，逐渐在各种行业得到了应用。在多媒体技术应用过程中，主要的实现途径包括基于 PC 上的实现，基于 ASIC 的实现，基于嵌入式系统的实现三种方式。基于 PC 上实现主要利用 PC 丰富的软硬件资源，可以快速的建立应用系统，这种方式易于实现；而基于 ASIC 的应用，比较适合大规模的应用，可以降低这种应用系统的成本；而嵌入式系统的实现介于二者之间，可以在比较丰富的软件资源的基础上编程实现多媒体应用，但是，嵌入式系统的开发对于开发工程师的要求比较高，不但需要知道系统开发的软硬件知识外，还要了解多媒体技术的知识细节，这些要求也一直是嵌入式多媒体系统开发的一个很大的难点。

DSP 作为一种运算能力相当出色的 CPU，在嵌入式多媒体处理的应用开发中占有重要的地位。TI 的 C6000 系列 DSP 以其独特的体系结构和强大的运算能力，十分适合在多媒体的音视频处理中应用。当前在嵌入式多媒体处理系统的开发过程中，除了系统的软硬件开发外，开发者还要注重考虑在 C 代码的基础上多媒体技术的具体实现，针对 DSP 特有的体系结构，在汇编层面对系统进行优化的，这样才能体现出 DSP 相对于其他处理器的优点，发挥 DSP 在多媒体处理方面的强大的能力。因此在系统的开发过程中，要求既对多媒体的算法有相当的了解，又对 DSP 的低层体系结构和汇编语言比较熟悉，这对开发工程师来说是一个非常大的挑战。而且，在多媒体系统开发过程中存在很多的同样的工作，在每次开发一个系统的时候，都需要重复很多复杂而又艰难的工作，耗费大量人力物力，而且开发时间也很长，例如在 MPEG1,MPEG2,MPEG4,H263,H264 等各种编解码标准中，有很多函数模块都是几乎一样的，如块的 DCT 变换函数,还有宏块求 SAD 函数等等。本文根据各多媒体标准和常用的多媒体操作，提出各类通用的函数，并在 TI C6000 平台上予以优化，开发出多媒体处理函数库，这样可以在开发过程中自由调用从而充分发挥 DSP 处理能力，而且缩短多媒体系统开发时间。由于多媒体处理涉及面广,包括了音频、视频、图像、图形、文字等等的处理,整个函数库比较庞大,本文则主要研究视频编码部分,下面对多媒体处理特别是编码

的技术及编码的标准发展进行阐述。

1.2 音视频压缩标准及发展

1.2.1 压缩编码技术

多媒体信息主要包括文字、声音、图像、图形、和视频等内容。各种媒体信息，特别是图像和动态视频，数据量非常之大。例如：一幅 640x480 分辨率的 24 位真彩色图像的数据量约力 900kb；一个 100Mb 的硬盘只能存储约 100 幅静止图像画面。显然，这样大的数据量不仅超出了计算机的存储能力，更是当前通信信道的传输速率所不及的。因此，为了存储、传输这些数据，必须进行压缩。所谓压缩，就是去除信息中的相关性，也即冗余，使得用更少的介质能够存储和传输更多的信息。视频中的冗余包括图像画面中相邻的象素之间的相关性造成的空间冗余，运动图像中前后帧图像之间的相同的背景和对象而造成的时间冗余，实际平均码长与信息熵之间的差距造成的编码冗余，因人眼视觉非均匀性，可以去除而不引起主观质量下降的视觉冗余，以及一些先验知识造成的知识冗余。

压缩编码技术从压缩前后是否有信息损失来分，有无损压缩和有损压缩两类。无损压缩指对压缩后的数据进行还原，解压缩后的数据与原来的数据完全相同。一般用于要求重构的信号与原始信号完全一致的场合。无损压缩算法主要特点是压缩比较低，为 2:1~5:1，一般用来压缩文本数据。由于压缩比的限制，仅使用无损压缩方法不可能解决图像和数字视频的存储和传输问题。有损压缩是指对使用压缩后的数据进行重构，重构的数据与原来的数据有所不同，但不影响人对原始资料表达的信息造成误解。有损压缩算法主要特点是压缩比高，为几十到几百倍。一般用于图像，声音，视频压缩。

在多媒体应用中常用的具体压缩方法有：脉冲编码调制、统计编码、预测编码、变换编码、混合编码，这些编码方法都广泛的应用于多媒体压缩标准中。统计编码是指根据消息出现概率的分布特性而进行的压缩编码。其中典型的算法有行程编码 Huffman 编码、LZW 编码、算术编码等。预测编码的算法是先对原始模拟信号作脉冲取样，把实际样值与预测样值之间的差进行量化。解压时，也用同样的预测器，把预测出的值与已存储的量化后差值相加，产生近似的原始信号。变换编码是指先对信号进行某种函数变换,从一种信号空间变换到另一种信号空间，再对变换后的信号进行编码。混合编码则是使用两种或两种以上的编码方法

混合进行编码称为混合编码，能提高数据压缩的效率。例如：多媒体图像压缩标准中都采用混合编码如 JPEG，MPEG 等。

1.2.2 视频压缩标准

视频压缩是多媒体领域中的重要内容，自上世纪 80 年代，针对视频会议、网络通讯、数字广播等广泛的应用场合制定了一系列的标准。国际上主要的视频编解码标准有两大系列：国际标准化组织和国际电工委员会第一联合技术组（ISO/IEC JTC1）制定的 MPEG 系列标准；ITU 针对多媒体通信制定的 H.26x 系列视频编码标准[7]。此外，伴随着视频编解码技术的进步和标准的更新换代，一些组织又提出了方案更简洁，知识产权政策更明晰的视频标准 AVS。伴随着多媒体技术标准化发展的过程中，一些公司也提出了自己的标准。

从最早的 H.261 视频编码提案，经过 H.262、H.263、MPEG1、MPEG2 以及现在不断发展的 MPEG4 等等视频编码标准，他们都有一个共同的目标，就是实现在尽可能低的码率情况下获得尽可能高的图像质量。这也促使 ISO/IEC 和 ITU 两大国际标准化组织联合起来制定了新一代视频编码标准 H.264。下面简要回顾一下主要视频编码标准的发展历程如图 1.1。

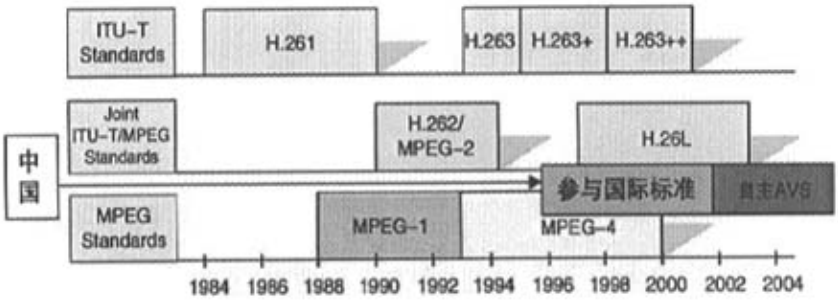


图 1音视频标准发展史

图 1.1 音视频标准发展史

H.261 颁布于 1990 年，是最早出现的视频编码提案，目的是用于将网络视频会议和可视电话业务等的视频编码技术纳入到一个规范中，标准的输入的图像为 CIF(352x288)或者 QCIF(176x144)，帧率小于 30，输出码率为 Px64kbps，其中 1<P<30，在 ISDN 信道上最高

传输率为 19.2Mbps，足以传输 VHS 质量的视频信号。H.261 的编码方案是基于运动补偿，帧间预测，和 DCT 的混合编码算法。将每帧图像分成 8x8 的子块，再组成宏块、块组，宏块由 4 个 8x8 的亮度块和 2 个 8x8 的色度块组成，每个块组由 3x11 个宏块组成，形成一个多层次的块组结构。H.261 标准大体上分为两种编码模式：帧内模式和帧间模式。对于缓和运动的人头像，帧间编码模式将占主导位置；而对画面切换频繁或运动剧烈的序列图像，则帧间编码模式要频繁地向帧内编码模式切换。以后的视频编码标准也以此为基础，不断加入一些新的技术。

1993 年 ISO/IEC 制定了 MPEG-1 标准，它是针对 1.5Mbit/s 速率的数字存储媒体运动图像及其伴音编码制定的国际标准，该标准的制定使得后来的基于 CD-ROM 的数字视频等产品成为可能。标准的输入图像格式为 352x240x30 或者 352x288x25，输出码率为 1.5M，其中 1.1Mbit/s 用于视频。MPEG-1 的编码方案，同样是基于运动补偿，帧间预测，和 DCT 的混合编码算法。为了追求高的压缩效率，去除图像序列的时间冗余度，同时满足多媒体等应用所必须的随机存取要求，MPEG-1 视频把图像编码分成 I 帧、P 帧、B 帧和 D 帧共 4 种类型。I 帧为帧内编码帧，编码时采用帧内 DCT 编码；P 帧为预测编码帧，采用前向运动补偿预测和误差的 DCT 编码，由其前面的 I 或 P 帧进行预测；B 帧为双向预测编码帧，采用双向运动补偿预测和误差的 DCT 编码，由前面和后面的 I 或 P 帧进行预测；D 帧为直流编码帧，只包含每个块的直流分量。

1995 年 MPEG 组织推出的 MPEG-2 标准是在 MPEG-1 标准基础上的进一步扩展和改进，主要是针对数字视频广播、高清晰度电视和数字视盘等制定的 4～9Mbit/s 运动图像及其伴音的编码标准，MPEG-2 是数字电视机顶盒与 DVD 等产品的基础。为了更具权威性，ISO 把 MPEG2 标准提交给国际电信联盟纳入 ITU 的 H 系列标准中，因此，现在 MPEG2 视频压缩部分又称为 ITU/H.262 标准。MPEG-2/H.262 标准采用的核心技术还是分块 DCT 和帧间运动补偿预测技术，主要有以下几个方面的扩展：

（1） 输入/输出图像彩色分量之比可以是 4∶2∶0，4∶2∶2，4∶4∶4。

（2） 输入/输出图像格式支持 352x288～1920x1152 之间的任何图像。

（3） 专门设置了按帧编码和按场编码两种方式。

（4） 在空间分辨率、时间分辨率、信噪比方面的可分级。

（5） 码流结构的可分级性。

（6） 输出码率可以是恒定的也可以是变化的， 以适应同步和异步传输。

1996 年 3 月 ITU-T 基于 ITU-T 的 H.261 标准制定了 H.263 标准，它是一种用于低比特

率视频业务中运动图像部分的压缩编码方法。标准输入的格式支持 CIF(352x288)，QCIF(176x144)，SUBQCIF（128x96），4CIF（704x576），16CIF（1408x1152）。视频编码算法的基本思想与 H.261 相比，H.263 标准采用了半象素精度位移估计。除了基本的视频源编码算法外，为了改善性能，它包含 4 个可选的编码方案：非限制运动矢量，先进预测模式，PB 帧模式和基于语法的算术编码。

2000 年底 MPEG-4（ISO/IEC14496）正式成为国际标准。MPEG-4 制定的初衷是针对视频会议，视频电话的极低码率编码。但为了适应多媒体传输、存储、检索等不同的应用需求，最终制定了现在意义上的基于对象的压缩编码标准。标准输入格式支持大于 QCIF（176x144）的任何分辨率的图像。MPEG-4 与 MPEG-1 和 MPEG-2 标准区别在于它是基于内容的压缩编码方法，它对一幅图像按内容切分为块，将感兴趣的物体从场景中分割出来进行编码，可以获得高压缩比效果，而且可以支持基于内容的交互。MPEG-4 引入视频对象 VO(Video Object)和视频对象平面 VOP(Video Object Plane)概念来表示内容。视频对象 VO 的构成依赖于具体的应用和实际系统所处的环境。VO 的描述通过三类信息来实现：运动信息、形状信息和纹理信息[2]。

H.264 是 ITU-T 和 ISO/IEC 的 MPEG 的联合视频组（JVT）开发的一个新的数字视频编码标准，它既是 ITU-T 的 H.264，又是 ISO/IEC 的 MPEG-4 的第 10 部分。H.264 同样是基于传统的混合编码系统，在局部采用了一系列的技术，使得在相同的重建图像质量下，能够比 MPEG-4 VISUAL 节约 50%左右的码率。

H.264 的主要技术亮点包括：

（1） 视频编码层（VCL）和网络提取层（NAL）结构

（2） 支持 1/4 或 1/8 像素精度的运动矢量。

（3） 4×4 块的整数变换

（4） 提供了标准的 UVLC 和 CABAC 熵编码

（5） 先进的帧内预测模式

（6） 面向 IP 和无线环境

2002 年 6 月，我国成立了 AVS 工作组，并于 2003 年 12 月定稿 AVS 视频部分。AVS 是我国自主制定的音视频编码技术标准，以当前最先进的 AVC/H.264 框架为基础，强调自主知识产权，同时充分考虑了实现的复杂程度。相对于 H.264，AVS 主要的特点有：8x8 的整数变换与 64 级量化；亮度和色度帧内预测都是以 8x8 块为单位，亮度预测采用 5 种预测模式，色度块采用 4 种预测模式；采用 16x16，16x8，8x16，8x8 四种块模式进行帧间运动

补偿；在 1/4 象素运动估计方面，采用不同的四抽头滤波器进行半象素插值和 1/4 象素插值；P 帧可以利用最多 2 帧的前向参考帧，而 B 帧可采用前后各一个参考帧。在高分辨率应用中，其压缩效率明显比数字电视，存储媒体中的 MPEG-2 高。同时，在压缩效率相当的前提下，其实现的复杂度有比较 H.264 的 main profile 大为降低[15][16][17][18]。

1.2.3 音频压缩标准

由于数字音频压缩技术具有广阔的应用范围和良好的市场前景，因而一些著名的研究机构和大公司都不遗余力地开发自己的专利技术和产品，这些音频压缩技术的标准化工作就显得十分重要。ITU-T 在语音信号压缩的标准化方面做了大量的工作，制订了如 G.711、G.721、G.728 等标准，并逐渐受到业界的认同。 在音频压缩标准化方面取得巨大成功的是 MPEG 系列标准，由于在制订标准时对许多压缩技术进行了认真的考察，并充分考虑了实际应用条件和算法的可实现性（复杂度），得到了广泛的应用。

在语音编码方面，大致可以分为三类：波形编码、参数编码和混合编码。波形编码算法比较简单，直接对语音信号形成的波形进行处理和加工，方法简单，音质优良，不过码率较高。参数编码根据人的发声机理，从语声的波形中提取表征声道和声源激励的有关特征参数，再利用这些特征参数通过模型合成出语音信号，这类编码算法复杂，合成语音质量有所下降，但是码率较低。混合编码是将波形编码和参数编码的原理结合起来，音质比较好。伴随着语音应用需求和编码技术的发展，ITU-T 制定的一系列的标准也是采用这些技术实现的，如 G. 711 采用的是采用脉冲编码调制编码方式，G. 726 采用的是自适应差分脉冲编码调制编码方式，而 G. 728 是以低时延码激励线性预测编码为基础的。

在音频编码方面，主要的音频编码方式包括 MPEG 系列中的音频编码和 DOLBY 实验室推出的 AC 系列编码标准。MPEG 系列音频标准中，无论是 MPEG1，MPEG2，MPEG4 都有 3 层组成，其中 MP3 作为 MPEG 音频标准的第 3 层应用最为广泛。MP3 编码算法主要包括时频映射，位分配及量化编码，帧形成，心理声学模型几个部分。编码首先采用正交滤波器组，将 20kHz 左右的信号划分成相等的 32 个子带，然后对子带样值作 MDCT 以补偿子带滤波的不足，同时采样值通过心理声学模型计算出个频带的掩蔽阀值，通过掩蔽阀值选择量化步长对采样值进行量化，最后将各种信息以及霍夫曼码打包成比特流。在 MP3 格式定稿之后，MPEG 又提出了更高质量的音频编码，简称 AAC。AAC 在 MP3 的基础上，增加了瞬时噪音整形，长时预测，频域预测等技术，达到更好的编码效果。AC 系列音频编码

补偿；在 1/4 象素运动估计方面，采用不同的四抽头滤波器进行半象素插值和 1/4 象素插值；P 帧可以利用最多 2 帧的前向参考帧，而 B 帧可采用前后各一个参考帧。在高分辨率应用中，其压缩效率明显比数字电视，存储媒体中的 MPEG-2 高。同时，在压缩效率相当的前提下，其实现的复杂度有比较 H.264 的 main profile 大为降低[15][16][17][18]。

1.2.3 音频压缩标准

由于数字音频压缩技术具有广阔的应用范围和良好的市场前景，因而一些著名的研究机构和大公司都不遗余力地开发自己的专利技术和产品，这些音频压缩技术的标准化工作就显得十分重要。ITU-T 在语音信号压缩的标准化方面做了大量的工作，制订了如 G.711、G.721、G.728 等标准，并逐渐受到业界的认同。 在音频压缩标准化方面取得巨大成功的是 MPEG 系列标准，由于在制订标准时对许多压缩技术进行了认真的考察，并充分考虑了实际应用条件和算法的可实现性（复杂度），得到了广泛的应用。

在语音编码方面，大致可以分为三类：波形编码、参数编码和混合编码。波形编码算法比较简单，直接对语音信号形成的波形进行处理和加工，方法简单，音质优良，不过码率较高。参数编码根据人的发声机理，从语声的波形中提取表征声道和声源激励的有关特征参数，再利用这些特征参数通过模型合成出语音信号，这类编码算法复杂，合成语音质量有所下降，但是码率较低。混合编码是将波形编码和参数编码的原理结合起来，音质比较好。伴随着语音应用需求和编码技术的发展，ITU-T 制定的一系列的标准也是采用这些技术实现的，如 G. 711 采用的是采用脉冲编码调制编码方式，G. 726 采用的是自适应差分脉冲编码调制编码方式，而 G. 728 是以低时延码激励线性预测编码为基础的。

在音频编码方面，主要的音频编码方式包括 MPEG 系列中的音频编码和 DOLBY 实验室推出的 AC 系列编码标准。MPEG 系列音频标准中，无论是 MPEG1，MPEG2，MPEG4 都有 3 层组成，其中 MP3 作为 MPEG 音频标准的第 3 层应用最为广泛。MP3 编码算法主要包括时频映射，位分配及量化编码，帧形成，心理声学模型几个部分。编码首先采用正交滤波器组，将 20kHz 左右的信号划分成相等的 32 个子带，然后对子带样值作 MDCT 以补偿子带滤波的不足，同时采样值通过心理声学模型计算出个频带的掩蔽阀值，通过掩蔽阀值选择量化步长对采样值进行量化，最后将各种信息以及霍夫曼码打包成比特流。在 MP3 格式定稿之后，MPEG 又提出了更高质量的音频编码，简称 AAC。AAC 在 MP3 的基础上，增加了瞬时噪音整形，长时预测，频域预测等技术，达到更好的编码效果。AC 系列音频编码加了瞬时噪音整形，长时预测，频域预测等技术，达到更好的编码效果。AC 系列音频编码

标准是 MPEG 系列外的另一个重要的标准。目前最重要的标准是 AC3，广泛应用于家用，影院等娱乐作品中。AC3 系统总共包括左，中，右，左环绕，右环绕，低音效果声 6 个声道，具体操作时，首先要对各声道信号进行频谱分析，然后滤波压缩，频谱包络编码，再进行比特率的重新分配，重组 AC3 数据流[21][22]。

1.2.4 静态图像压缩及其他

静态图像的压缩和处理也是多媒体技术的重要组成部分之一，压缩标准主要有 JPEG 和 JPEG2000。ISO/IEC 联合图像专家组制定的静止图像压缩标准 JPEG 是适用于连续色调（包括灰度和彩色）静止图像压缩算法的国际标准。JPEG 算法共有 4 种运行模式，其中一种是基于空间预测（DPCM）的无损压缩算法，另外 3 种是基于 DCT 的有损压缩算法。无损压缩算法，可以保证无失真地重建原始图像；基于 DCT 的顺序模式，按从上到下，从左到右的顺序对图像进行编码，称为基本系统；基于 DCT 的递进模式，指对一幅图像按由粗到细对图像进行编码；分层模式是指以各种分辨率对图像进行编码，可以根据不同的要求，获得不同分辨率的图像。目前主要采用的是基于 DCT 的顺序模式，广泛应用于网络，数码产品中。与以往的 JPEG 标准相比，JPEG-2000 压缩率比 JPEG 高约 30%，它有许多原先的标准所不可比拟的优点。JPEG-2000 与 JPEG 最大的不同，在于它放弃了 JPEG 所采用的以 DCT 变换为主的分块编码方式，而改为以小波变换为主的多分辨率编码方式。而且，JPEG-2000 能实现无损压缩（lossless compression）。在实际应用中，有一些重要的图像，如卫星遥感图像、医学图像、文物照片等，通常需要进行无损压缩[44]。JPEG-2000 还有一个很好的优点就是误码鲁棒性好。因此使用 JPEG-2000 的系统稳定性好，运行平稳，抗干扰性好，易于操作。JPEG-2000 能实现渐进传输，这是 JPEG-2000 的 一个极其重要的特征，这在网络传输中具有非常重大的意义。JPEG-2000 另一个极其重要的优点就是感兴趣区特性。用户在处理的图像中可以指定感兴趣区，对这些区域进行压缩时可以指定特定的压缩质量，或在恢复时指定特定的解压缩要求，这给人们带来了极大的方便。

数字图像处理是一个目前相当热门的研究领域，在多媒体处理领域占的地位越来越大。图像处理除了视频压缩以外，还有很多的其他方面，包括图像变换，图像增强和复原，图像分割，图像描述，图像分类。对于图像变换由于图像阵列很大，直接在空间域中进行处理，涉及计算量很大。因此，往往采用各种图像变换的方法，如傅立叶变换、沃尔什变换、离散

余弦变换等间接处理技术，将空间域的处理转换为变换域处理，不仅可减少计算量，而且可获得更有效的处理。目前新兴研究的小波变换在时域和频域中都具有良好的局部化特性，它在图像处理中也有着广泛而有效的应用。图像增强和复原的目的是为了提高图像的质量，如去除噪声，提高图像的清晰度等。图像增强不考虑图像降质的原因，突出图像中所感兴趣的部分。如强化图像高频分量，可使图像中物体轮廓清晰，细节明显；如强化低频分量可减少图像中噪声影响。图像复原要求对图像降质的原因有一定的了解，一般讲应根据降质过程建立"降质模型"，再采用某种滤波方法，恢复或重建原来的图像。图像分割是数字图像处理中的关键技术之一。图像分割是将图像中有意义的特征部分提取出来，其有意义的特征有图像中的边缘、区域等，这是进一步进行图像识别、分析和理解的基础。虽然目前已研究出不少边缘提取、区域分割的方法，但还没有一种普遍适用于各种图像的有效方法。因此，对图像分割的研究还在不断深入之中，是目前图像处理中研究的热点之一。图像分类（识别）属于模式识别的范畴，其主要内容是图像经过某些预处理（增强、复原、压缩）后，进行图像分割和特征提取，从而进行判决分类。图像分类常采用经典的模式识别方法，有统计模式分类和句法（结构）模式分类，近年来新发展起来的模糊模式识别和人工神经网络模式分类在图像识别中也越来越受到重视。

1.3 多媒体处理系统方案比较

多媒体处理有一个很大的特点，那就是需要很大的运算量和存储容量，这也给多媒体处理系统实现带来了很大的挑战，因此多媒体处理芯片的开发一直是各大半导体公司研究开发的热点。目前，主要的多媒体处理系统主要有 3 种实现方法：基于通用 CPU 的 PC 实现，基于 ASIC 的嵌入式系统，基于通用 DSP 的嵌入式系统实现。

基于通用的 PC 实现是利用当前的 PC 丰富的硬件和软件资源，特别是 Intel 的 MMX 提供了较完整的多媒体指令集和流水线，可以提供较强的多媒体处理能力，PC 的内存容量大，可以方便的存储大量的多媒体数据，而且 PC 的外围设备众多，扩展方便，可以容易搭建起系统方案，由于软件功能强大，不需要很大的硬件开销。但是 PC 的缺点也很明显，由于多媒体运算量大，所以要占用几乎所有的 CPU 处理能力，从而导致 PC 不能完成其他任何任务，另外 PC 的体积大，功耗大也限制了这类多媒体应用 。

直接采用硬件 ASIC 处理的优点是方便集成，系统集成方便，不需要软件的开发，开发

周期短，而且由于 ASIC 的产量大[53]，从而大大降低了系统的成本，此外，ASIC 体积小，功耗小，很适合在嵌入式系统中应用。但是，由于 ASIC 的所有功能都固化在硬件上了，所以只能应用在专一的场合，应用对象范围小，系统升级和修改的代价相当的高，对特殊环境缺乏应变力。

通用 DSP 实现多媒体系统是目前应用最为广泛的方案之一。随着 DSP 向高速化，低功耗，多媒体化，多处理器的方向发展，使得 DSP 应用更加方便，质量更加好。通用 DSP 平台主要有以下几个优点：

（1）用户开发自由度大，支持各种灵活的方案。由于 DSP 的可编程特点，DSP 系统可以应用于很多不同的环境中，大大增加了系统的应用范围。

（2）可以在最快的时间内满足市场的需求，而且可以在第一时间里提高产品的性能，升级方便。

（3）由于 DSP 特有针对多媒体的体系结构和指令集，所以具有强大的处理能力，可以在单片的 DSP 上完成大量的多媒体运算。

（4）外围接口丰富，可以直接于视频输入输出进行连接，不需要其他的板卡支持，满足各种系统的需求，可以通过网络接口直接和网络连接，构成网络产品。由于 DSP 体积小可以很容易扩展成各种系统级的板卡。

（5）芯片功耗小，提高了产品的稳定性。

DSP 系统的优点众多，但也对开发者提出了较高的要求，对硬件和软件算法都要求有比较深入的了解，才能开发出优秀的产品[45][46][47]。基于以上分析，本文提出了基于 TI DSP 上多媒体处理库如下。

1.4 MML 多媒体处理库

多媒体处理库见图 1.2 包括很多领域，图像处理，信号处理，图像识别的具体操作，视频处理等等。多媒体处理库包括了几乎各种多媒体计算任务和各种多媒体数据类型。各个领域特别是视频编解码中几乎所有的需要消耗大量时间的函数都进行了优化。这些函数在多媒体应用中可以频繁使用。这种将处理函数包含在一个数据库中的一个明显的优点就是提高编程的效率，从一个项目到另一个项目的开发只需要采用不同的函数即可。

具体的底层函数分为以下几个部分：

（1） 信号处理和音频处理

（2） 图像处理和视频处理

（3） 矩阵算术和几何变换

（4） 编码运算

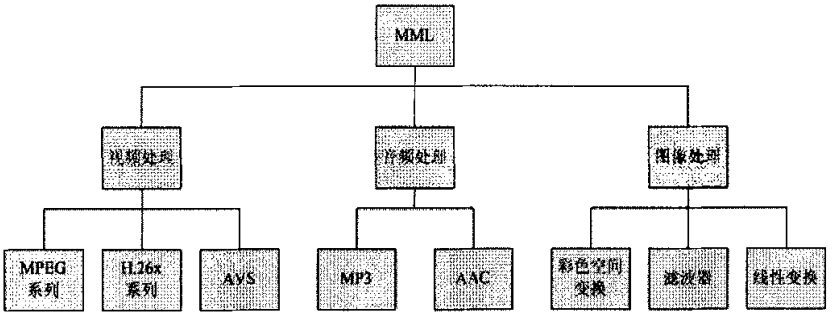


图 1.2 多媒体处理库层次图

函数库的另外一个特点就是基于这些低层优化的函数，可以构建高层的多媒体应用函数。在应用的过程中，可以根据需求的不同采用更低层的函数或者相对高层的函数。也可以引用各层函数完成整个软件的设计。

当前在多媒体系统的开发过程中，开发者总是注重算法层面上考虑的一些优化技术达到有优化的目的，这些都是在 C 代码的基础上的，C 代码不能够发挥 DSP 特有的体系结构的优势，无法发挥 DSP 在多媒体处理方面的强大的能力。本文在提出的多媒体处理库的基础上，使用各种优化技术，对各层次函数的予以优化，使这些函数在系统中运行的效果最佳。

1.5 本文的研究内容

重点研究和解决了 2 方面的问题，第一个研究了在底层函数优化的各种方法，诸如软件流水，线性汇编增大函数循环的处理的象素个数，软件流水，选择合适的汇编指令等技术，并对各底层函数进行了优化。第二就是对整个编码器的结构和 DM642 的缓存体系进行了分析，在编码器层安排合适的存储体系和 EDMA 的数据流组织，完成上层函数的优化。另外还调整了部分编码器算法结构以适合在 DM642 上运行。

本文在第一章首先介绍了目前多媒体处理的技术方法发展概况，然后简单介绍了一下多

媒体处理系统实现的平台和多媒体处理库的概念。然后在第二章详细介绍了一下多媒体处理库的结构和具体的函数组成，以及具体的应用领域。在第三章具体介绍了一下 DSP 优化平台的特点，具体硬件的体系结构，这些都是函数优化的基础，函数优化的方向正是充分发挥这些硬件的能力。在介绍完函数库和硬件平台以后，在第四章详细介绍了各层次函数的各种优化方法，以及实现的过程，这也是本文的重点和主要工作。

第二章 多媒体处理库(MML)

根据多媒体应用的需要，我们定义了 MML 的框架，主要包括如下部分：视频编解码库，音频处理库，图像处理库等，其中视频编码库作了重点研究.

2.1 视频处理库

MML 视频处理库目前主要包括关于 MPEG4 和 H.264 标准的视频编解码函数。对于库函数的组成，我们作了分层处理，具体包括四层函数如图 2.1 所示。视频库包括编解码器层，帧层，功能模块层，底层等四个层，高层的函数可以调用低层的函数来实现，低层的函数可以级联起来组成高层的函数。如果上层的编解码不能符合系统的要求，开发工程师可以直接调用上层的编解码器直接构成自己的系统，否则，开发工程师可以调用低层的函数来搭建满足自己系统要求的编解码器，另外还可以调用低层的函数完成自己的系统中其他的算法。

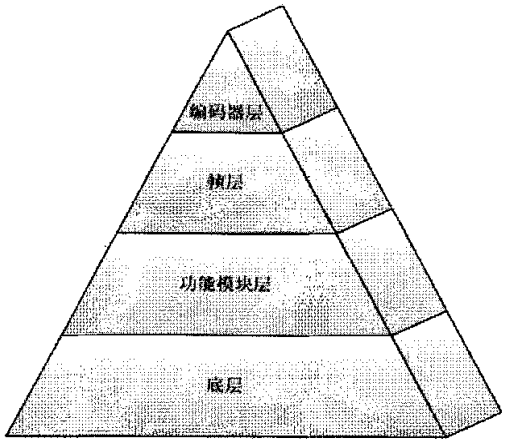


图 2.1 MML 视频编码器层次

A．编码器层：这一层的函数 MML_Encoder(Configuration *)是为不熟悉视频标准的用户准备的。用户完全不需了解视频编解码的流程和实现方法，只要将必需的参数初始化后调用该层的函数，就可方便实现视频编解码的全过程。为了达到这个目的，需要为 MML_Encoder 函数构建一个包含所有编码必需的参数的结构体类型 Configuration，并将一个指向它的结构体指针作为 MML_Encoder 函数的入口参数。它的成员包括：指向源文件首

地址的指针*SourceFile，指向目的码流首地址的指针*OutStream，编码图像的尺寸FrameHeight 和 FrameWidth，待编的帧数 FrameNumber，以及量化参数 QuantParameter 等等。用户根据自己的需要给这些成员赋上初值后，就可以调用 MML_Encoder 函数进行编码了。

B．帧层：这层的函数包括 MML_EncodeI、MML_EncodeP 和 MML_EncodeB。编码器层的函数可能会让一些用户觉得缺乏灵活性，因此，逐步开放用户自定义程序的空间是很必要的。这些服务于帧级的函数可以帮助用户方便快捷地搭建起一个编码器。这几个函数体需要传入的参数包括：指向存放头信息的结构体指针*Header，当前待编码帧的 Y、U 和 V 数据的首地址*CurrentY，*CurrentU 和*CurrentV，以及指向目的码流的指针*StreamPointer。

C．功能模块层：目前被广泛采用的基于块匹配的混合编码方法是将编码帧划分为 NxN 的块，对每一个块进行相对独立的处理。其核心思想是利用帧间预测编码消除图像序列中的时域冗余，利用变换编码来消除频域冗余。为了进一步增强函数的开放性，的各个功能模块都可以被封装起来。

D．底层：选择构建底层函数的模块的依据有两条：首先，它必须是标准中技术相对固定的部分，而且经常需要使用它们来构建高层语法，例如 SAD 计算和插值；其次，由于建立函数库的目的不仅仅在于让用户节省编写底层代码的时间，还要为提升用户视频标准代码的执行效率服务，所以还应该选择那些运算量大的技术部分将它们封装成模块，在提高它们的运行速度后提供给用户使用。

另外，为给用户提供更多的方便，还可以设计一些介于以上四个层次之间的函数。例如，MML_FullSearch（全搜索）、MML_DiamondSearch（菱形搜索）以及其他搜索函数可以用来构建功能模块层的 MML_MotionEstimation 函数，但它们又需要通过调用底层的 SAD 函数等来实现。链路中有些模块即相互独立，又紧密耦合，一个函数的输出数据会马上提供给下一个函数作为输入。例如，残差经过变换后的数据即是量化模块的操作对象。为了节省这些数据在函数间传递的时间，以达到更好的优化效果，可以将这些紧密连接的函数合并组成一个较大的函数。

以下为视频处理库的部分主要函数：

2.1.1 视频编解码器层

编解码器层目前主要包括 MPEG4 和 H.264 编解码器。函数将所有的细节封装在函数内部，用户根本不用了解编解码器内部的操作原理，只需要将编码方式通过编解码器结构体传

如函数中即可。函数主要包括

（1）MML_Encoder_MPEG4，

（2）MML_Decoder_MPEG4，

（3）MML_Encoder_H264，

（4）MML_DecodeI_H264

在引用函数时，编码层函数可以将原始图像序列编码为对应的标准的码流。解码层函数则将码流解码为原始图像序列。

2.1.2 编解码器帧层

编码帧层函数主要是对一帧图像完成 MPEG4 和 H.264 的标准编码，输出该帧图像对应的码流。MPEG4 和 H.264 帧编码主要有 I 帧，P 帧，B 帧 3 种编码方式。引用函数只需要对每一帧原始图像输入到帧编码器，输出为对应的码流。主要函数包括：

（1） MML_EncodeI_MPEG4，MML_EncodeP_MPEG4，MML_EncodeB_MPEG4

（2） MML_EncodeI_H264，MML_EncodeP_H264，MML_EncodeB_H264

编码帧层函数主要是对一帧图像码流完成 MPEG4 和 H.264 的标准解码，输出该码流对应的图像。MPEG4 和 H.264 帧编码主要有 I 帧，P 帧，B 帧 3 种编码方式。对应的 6 种码流可以分别对应的解码函数。引用函数只需要对每一帧图像对应的码流输入到帧解码器，输出为码流对应的图像。主要函数包括：

（1） MML_DecodeI_MPEG4 ，MML_DecodeB_MPEG4，MML_DecodeP_MPEG4

（2） MML_DecodeI_H264，MML_DecodeP_H264，MML_DecodeB_H264

2.1.3 编解码器功能模块层

功能函数主要包括编解码器中对宏块或者块的处理函数，包括变换，量化，MC，ME 函数。

（1）变换函数

MML_ DCT8x8FWD ，MML_ ICT8x8FWD

前者对二维帧缓存中的一个 8x8 块进行正向离散余弦变换，根据二维帧缓存的存储方式

的不同而选择不同的函数，应用于 MPEG4 的编解码器中。后者对当前 4x4 块中的像素值进行正向整数变换，应用于 H.264 编解码器中。

MML_ DCT8x8INV，MML_ ICT8x8 INV

前者对当前 4x4 块中的 ICT 系数进行反向整数变换。后者对二维帧缓存中的一个 8x8 块进行反向离散余弦变换，根据二维帧缓存的存储方式的不同而选择不同的函数。

（2）量化函数

MML_QuantIntra_MPEG4 ，MML_QuantInter_MPEG4 ，MML_Quant_H264

第一个函数对除 DC 系数之外的其他 63 个 DCT 系数进行量化，并用 pCount 标记最后一个非零系数的位置，以方便以后的操作。第二个函数对所有 64 个 DCT 系数进行量化，并用 pCount 标记最后一个非零系数的位置，以方便以后的操作。第三个函数对 4x4 块的 ICT 系数进行量化，并用 pCount 标记最后一个非零系数的位置，以方便以后的操作。前两个函数应用于 MPEG4 编解码器中，第三个函数应用于 H.264 编解码器中。

MML_QuantInvIntra_MPEG4，MML_QuantInvInter_MPEG4 ，MML_QuantInv_H264

第一个函数对帧内块除 DC 系数之外的其他 63 个 DCT 系数进行反量化操作，而第二个函数对帧间块的 64 个 DCT 系数进行反量化，输出值被钳制在[-2048，2047]之间，并且在帧间反量化函数中，还要添加 mismatch control 处理。第三个函数对 4x4 块的 ICT 系数进行反量化。前两个函数应用于 MPEG4 编解码器中，第三个函数应用于 H.264 编解码器中。

（3）ME 函数

MML_MotionEstimation_16x16_MPEG4

该函数使用 SEA 方法对一个宏块进行运动搜索，其中包括 16x16 块和 8x8 块的精确到半像素的搜索，并给出宏块的最终编码模式，是帧间编码还是帧内编码，是采用一个 MV 还是四个 MV。同时，为方便码率控制，函数还计算了宏块残差的和。应用于 MPEG4 编码函数中。

MML_CMEOneMB_H264

MML_AdvancedIntraModeSelectOneMacroblock

Intra16x16SelectAndPredict

第一个函数对一个帧间宏块进行运动搜索，找出最优的帧间模式和各个块划分的运动矢量。第二个函数对一个帧内宏块进行 4x4 帧内预测，找出各个 4x4 块最优的帧内模式。第三

个函数对一个帧内宏块进行 16x16 帧内预测，找出最优的帧内 16x16 模式，并将相应的预测值存放在 pPredBuf 指向的地址中。这三个函数应用于 H.264 编解码中。

（4）MC 函数

MML_MotionCompensation_MPEG4

函数主要完成运动补偿，应用于 MPEG4 编解码器中。

（5）去块滤波

DeblockSegment

函数主要是对每个重建图像进行去块滤波，提高重建图像的质量，从而提高编码效率。

2.1.4 编解码底层函数

（1）SAD 函数

MML_SAD16x16，MML_SAD8x8

该函数将根据参考块和 MV 信息得到的预测基准和残差信息相加，得到重建块数据。其中的预测是基于半像素精度的，重建后的数据存放在 pDst 指向的地址中。

（2）求均值函数

MML_AverageMB_MPEG4，MML_AverageBlock_MPEG4

这两个函数将两个宏块/块的对应像素值逐一求平均，并将结果存放在 pDst 指向的地址中。它们被用于 B 帧的重建中。

（3）宏块重建类函数

MML_CopyMBHalfpel_MPEG4，MML_CopyBlockHalfpel_MPEG4

这两个函数将由 pSrc 和 pMV 确定的宏块/块数据复制到 pDst 所指向的地址中，过程是基于半像素精度的，它们被用于 P 帧和 B 帧的重建中。

（4）插值类函数

MML_InterpolateLuma_H264，MML_InterpolateChroma_H264

该函数采用六抽头滤波的方式对参考块进行插值。插值后的数据存放在 pDst 中。

（5）熵编码类函数

MML_EncodeMV_MPEG4

该函数对宏块的 MV 信息进行编码。

MML_Encode_IntraACVLC_MPEG4 ， MML_Encode_InterVLC_MPEG4

第一个函数被用来对帧内块的 DCT 系数进行编码。若 noDCcode == 0，从 pQDctBlkCoef[0]开始的 64 个系数被编码；否则，从 pQDctBlkCoef[1]开始的 63 个系数被编码。第二个函数被用来对帧间块的 DCT 系数进行编码，只采用之字形扫描方式。

MML_EncodeVLCZigzag_IntraDCVLC_MPEG4

MML_EncodeVLCZigzag_IntraACVLC_MPEG4

MML_EncodeVLCZigzag_Inter_MPEG4

前两个函数对帧内块系数进行之字形扫描，然后对其进行变长编码。它们的区别在于 IntraDCVLC 使用 Intra DC VLC 来对 DC 系数编码，而 IntraACVLC 使用 Intra AC VLC 来对 DC 系数编码。最后一个函数对帧间块系数进行之字形扫描，然后对其进行变长编码。

MML_EncodeCoeffsCAVLC_H264 ， MML_EncodeChromaDcCoeffsCAVLC_H264

第一个函数计算了对 4x4 块系数进行编码时必须的要素的值，包括 Trailing_One, Trailing_One_Signs, NumOutCoeffs, TotalZeros, pLevels 和 pRuns。第二个函数计算了对 2x2 色度 DC 系数进行编码时必须的要素的值，包括 Trailing_One, Trailing_One_Signs, NumOutCoeffs, TotalZeros, pLevels 和 pRuns。

MML_EncodeCoeffsCAVLC_H264 ， MML_EncodeChromaDcCoeffsCAVLC_H264

第一个函数计算了对 4x4 块系数进行编码时必须的要素的值，包括 Trailing_One, Trailing_One_Signs, NumOutCoeffs, TotalZeros, pLevels 和 pRuns。第二个函数计算了对 2x2 色度 DC 系数进行编码时必须的要素的值，包括 Trailing_One, Trailing_One_Signs, NumOutCoeffs, TotalZeros, pLevels 和 pRuns。

（6）熵解码函数

MML_DecodeVLC_IntraDCVLC_MPEG4 ， MML_DecodeVLC_IntraACVLC_MPEG4

第一个函数被用来对一个帧内块的 DC 系数进行解码。 第二个函数被用来对一个帧内块的 AC 系数进行解码。

MML_DecodePadMV_PVOP_MPEG4

该函数解码 P_VOP 中的帧间宏块的 MV 信息，并把解出的一个或四个 MV 存放到 pDstMVCurMB 中去。

MML_DecodeMV_BVOP_Forward_MPEG4

MML_DecodeMV_BVOP_Backward_MPEG4

MML_DecodeMV_BVOP_Interpolate_MPEG4

MML_DecodeMV_BVOP_Direct_MPEG4

这四个函数被分别用来解码 B_VOP 中前向模式，后向模式，双向模式，直接模式宏块的 MV 信息。

2.2 音频处理库

音频处理库主要是为了完成各种音频编码和语音编码的构成部分。音频处理库主要介绍数列处理类函数，滤波类函数，变换类函数，语音编码类函数，音频编码类函数等等。具体函数种类如下[4]：

数列处理类函数：

（1） 逻辑和移位函数

（2） 数学运算函数

（3） 数组转换函数

（4） 维特比解码函数

（5） 窗函数

滤波类函数：

（1） 卷积函数

（2） FIR 滤波函数

（3） 一阶 FIR LMS 滤波函数

（4） 多阶 FIR LMS 滤波函数

（5） IIR 滤波函数

（6） 中值滤波函数

转换类函数：

（1） 快速傅立叶变换函数

（2） 离散余弦变换函数

（3） 希尔伯特变换函数

（4） 小波变换函数

语音编码类函数：

（1）G．729 相关的函数

（2）G．723 相关的函数

（3）G．726 相关的函数

音频编码类函数：

（1） 频谱数据预量化函数

（2） 求量化参数函数

（3） 尾数转换量化函数

（4） 改进离散余弦变换函数

（5） 块滤波函数

（6） 频域预测函数

（7） 哈夫曼编码函数

（8） 矢量量化函数

（9） MP3 编码函数

（10） AAC 编码函数

2.3 图像处理库

图像处理库函数主要包括图像逻辑和算术运算类函数，图像颜色转换类函数，图像滤波类函数，图像线性变换类函数，图像几何变换类函数，图像小波变换类函数，图像压缩类函数等。

其中，具体函数如下：

图像逻辑和算术运算类函数：

（1） 图像算术运算

（2） 图像几何运算

图像颜色转换类函数

（1） 颜色空间转换函数

（2） 图像抽样函数

（3） 彩色图像到灰度图像的转化函数

（4） Gamma 矫正函数

图像滤波函数：

（1） 中值滤波函数

（2） 线性滤波函数

（3） 分离滤波器函数

图像线性转换类函数：

（1） 快速傅立叶变换函数

（2） 窗函数

（3） 离散余弦变换函数

图像几何变换类函数：

（1） 图像反转函数

（2） 图像旋转函数

图像压缩类函数：

（1） 量化函数

（2） DCT 函数

（3） 抽样函数

（4） Huffman 编码函数

（5） JPEG 编码函数

（6） JPEG2000 编码函数

2.4 本章小结

本章主要介绍了具体的函数库，视频编解码器库分层实现，开发工程师可以根据自己的需要，选择适合系统的函数。编解码器库主要分为编解码器层，编解码帧层，编解码功能模块层，编解码底层函数。高层函数可以有低层搭建起来。然后介绍了音频处理库和图像处理库的部分函数组成。

第三章 TI DM642 结构特点

3.1 用于多媒体处理通用 DSP

目前主要的多媒体处理通用 DSP 包括 Phlips 公司的 PNX 系列，TI 公司的 C6000 系列多媒体处理的 DSP，ADI 公司的 BLACKFIN 系列，另外还有一些公司推出的相应的产品。简单介绍如下：

PNX1300 系列 DSP 是 Philips 公司开发生产的比较经典的多媒体处理领域的 DSP。PNX-1300 系列芯片正在被大规模应用开始于视频监控产品中。在 PNX-1300 系列成功应用的基础上，Philips 于去年推出了性能更高的 PNX-1500 系列。作为 PNX-1300 系列的升级换代产品，PNX-1500 系列处理能力更高、性能更好。该产品于 2004 年下半年批量上市，成为了数字视频应用的新亮点。PNX-1500 还在 PNX1300 的基础上增加了许多功能，成为该系列产品的新亮点。增加的新功能有：网络接口、IDE 接口；提供了开发信息化家电和数字视频设备的主要接口；视频输出：提供 LED 高分辨输出、高清视频输出（1920x1080）；视频处理单元：视频滤波和 De-interlace 处理；2D 图形加速器：可以生成图形；内嵌看门狗电路并具有两个 Reset 管脚；提高设计的可靠性。

2002 年 ADI 公司推出了 Blackfin 系列 DSP，其中的 ADSP-21535 是一款合适的数字视频应用的 DSP，ADSP-21535 具有 600MHZ 的核内时钟，300MHZ 主频，一个 VP 口，但是没有预览通道，接口资源也很丰富，Blackfin 系列的 DSP 采用双 MAC 的结构具有正交的类似 RISC 的微处理器指令集，使单指令多数据和多媒体操作都引入单指令结构。这样的 DSP 芯片结构不但易于编程，可以快速的信号处理和多媒体的处理，而且方便的扩展 USB、PCI I/O、UART、SPORT 等接口。非常适合对视频读入，处理以及传输。ADI 最新的双核 ADSP-21561 也是专业视频处理 DSP 领域内不容忽视的好产品。但是相比较 Philips 和 Ti，ADI 的数字视频 DSP 的劣势在于能够支持 Blackfin 的第三方算法太少，这也是造成虽然 Blackfin 的产品非常有特点，但是应用面要远远小于前面两家公司的一个主要原因。

Equator 公司继 MAP-CA 芯片后推出了 BSP-15 芯片，主频有 256、300、350 和 400MHz，芯片具有两个视频输入口和音频输入口，一个视频输出口，可以支持 SDRAM 最大为 128MB。也是音视频方面处理应用比较合适的选择。

此外对于特定的多媒体应用，各个半导体公司都推出了自己的 ASIC 芯片。例如对于 MPEG-4 算法众多公司推出了 ASIC 方案，例如 WIS 公司的 GO7007、Intime 公司的 IME6400、

Vweb 公司的 VW2010、Toshiba 的 TC35280XB、以及 LSI Logic 公司和 ST Micro 等等。

C6000 系列多媒体处理 DSP 是 TI 公司开发的众多系列中专门为多媒体应用的一个系列产品，TI 公司是 DSP 芯片的行业老大，它的众多产品多年来一直统治着这个行业，已经深入应用到了电子信息行业各个领域中。2003 年 TI 发布了 TMS320DM64X 系列的多媒体 DSP 产品，2004 年下半年批量供货，产品一经面世得到了数字视频行业的强烈关注。

TMS320DM64X 系列是特别为音视频处理方面的应用而开发的定点 DSP，特别集成了 VIDEO 单元负责视频信号的输入输出，可以很方便的连接各种视音频编解码器件。TMS320DM64X 系列各个型号的 DSP 具有相同的 CPU 核，只是不同的外设配置。如 DM642 则有 3 个 VEDIO 单元。每个 Video 单元又分成 A、B 两个口，A/B 口可以分别处理一路视频采集，因此 DM642 最多可以处理 6 路视频采集数据（不带音频）。如果将 Video 单元配置成 Video out 方式，则只能在 A 口输出，B 口不可以，因此 DM642 最多可支持 3 路视频输出。如果同时处理音频，每一个视频单元可以处理两路立体声。DM642 芯片功耗 1.5W，支持 SDRAM 最大为 32MB，同时也具有网络接口。另外 TMS320DM642 的运算能力非常强劲，可实现 4-6 路 CIF 实时 MPEG4 编解码算法，或 1 路 2CIF 的 H.264 编解码运算。具有教大的片上内存，以及丰富的接口资源。

3.2 TI DM642 的主要特点

3.2.1 C6000 系列 DSP

1997 年，美国 TI 公司发布了新一代 DSP 芯片 TMSC6000 系列，主要应用于多媒体应用，包括定点系列和浮点系列。其中定点系列是 TMS320C62xx，浮点系列是 TMS320C67xx，二者相互兼容。2000 年 3 月，TI 发布了新的 C64xx 内核，主频 1.1GHz，处理速度接近 9000MIPS，总体性能比 C62xx 提高了 10~15 倍。其中 C6416 在 600MHz 主频下，只利用 50%的运算能力就可以同时运行单通道 MPEG-4 视频编码，单通道 MPEG-4 视频解码和单通道 MPEG-2 视频编码的处理。2002 年，TI 公司为了处理多媒体方面的应用，在 C64xx 核的基础上开发出了 DM642。为了满足下一代嵌入式系统对网络编解码器的需求，TI 公司在 2005 年底，发布了利用达芬奇技术的新一代多媒体片上系统 DM6446，主要有运算单元和控制单元两部分组成。

TI C6000 系列 DSP 最主要的特点是内核有 8 个运算单元和超长指令集结构，从而可以在一个时钟周期内为 8 个运算单元取 8 个指令。8 个指令之间相互独立，分别在各个处理单元独自运行。编译器在对汇编程序进行编译的过程中，决定代码中的哪些指令合成一个超长指令包。这种指令的并行安排是静态的，也就是说，所有的指令在编译的时候都是决定了的，无论什么时候执行，都是不变的。每 32bit 中的第一个比特决定本指令是属于上一个指令同样的指令包还是属于一个新的指令包。对于取指令时，指令包的大小是一定的，总是 256 比特，而在运行的时候，指令包的大小是可变的。因为当运算单元没有准备好下一次运算时，超长指令集控制单元就不为在下一个时钟为运算单元提供运算指令。这种可变的运行指令包正是 C6000 与其他的超长指令集体系结构之间的最大不同，这种可变的运行指令包的好处就是可以减少存储器的开销。

鉴于 DSP 面向的是数据密集型的应用，频繁的数据存取会降低系统的性能，所以在总线结构上，为了适应大量数据的读取和存储，C6000 摒弃了传统的冯诺依曼体系结构，采用了数据总线和地址总线分开的哈佛结构。哈佛结构有以下几个明显的特点：

（1）使用两个独立的存储器模块，分别存储指令和数据，每个存储模块都不允许指令和数据并存，以便实现并行处理；

（2）具有一条独立的地址总线和一条独立的数据总线，利用公用地址总线访问两个存储模块（程序存储模块和数据存储模块），公用数据总线则被用来完成程序存储模块或数据存储模块与 CPU 之间的数据传输；

（3）两条总线由程序存储器和数据存储器分时共用。

3.2.2 DM642

3.2.2.1 DM642 的 CPU

DM642 的 CPU 核主要有以下几部分组成。如图 3.1

（1）取指令单元

（2）指令分配打包单元

（3）译指单元

（4）两个数据通道，每个包括 4 个处理单元

（5）控制寄存器

（5）控制逻辑

（6）测试，调试，中断逻辑

取指令，指令分配单元和译指单元可以在一个指令周期内将 8 个指令送到 CPU 的处理单元中去。指令的处理主要有数据通道来完成，每个数据通道由四个处理单元和 32 个通用寄存器组成。控制寄存器是用来配置和控制 CPU 的操作的。

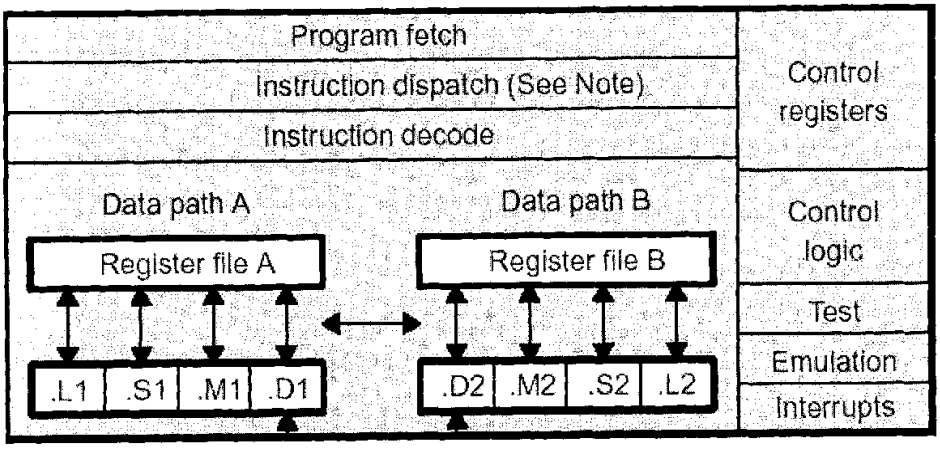


图 3.1 DM642 的 CPU 核结构图

DM642 有两个对称的数据通道如图 3.2，每个数据通道包括一组 32 个通用寄存器，4 个处理单元，两个存数据通道，两个取数据通道，两个寻址通道，两个寄存器交叉取数据通道。每组寄存器组可以用作数据寄存器，数据地址指针或者条件寄存器。每个寄存器可以存储多个 8 bit 的数据。对于大于 32 位的定点数或者 64 位的浮点数，可以存在寄存器对中。四个处理单元分别是 L，D，S，M。其中 L 单元负责算术运算和比较操作等运算；S 单元负责算术运算，逻辑处理，移位操作等运算；M 单元主要负责乘法运算，加罗斯域乘法；D 单元负责加减法操作和存取操作。每个运算单元在自己的数据通道中直接读写，也就是 L1，D1，M1，S1 单元读写寄存器组 A，L2，D2，M2，S2 单元读写寄存器组 B。DM642 提供了两个交叉通道使一个数据通道的处理单元可以访问另一个数据通道的寄存器组，其中 M，S，D 运算单元的只有第二个源操作数可以来自交叉数据通道，而 L 处理单元的 2 个源操作数都可以来自交叉数据通道。由于只有两个交叉数据通道，这就使每个数据通道在一个时钟周期内只能从另外一个数据通道的寄存器组中取一个源操作数。对于存储指令，DM642 可

以依次存储 64 比特的数据。因此实际上有四个 32 比特的取数据通道，通道 LD1a 存储数据通道 A 的低 32 位，LD1b 存取数据通道 A 的高 32 位，通道 LD2a 存储数据通道 B 的低 32 位，LD2b 存取数据通道 B 的高 32 位。

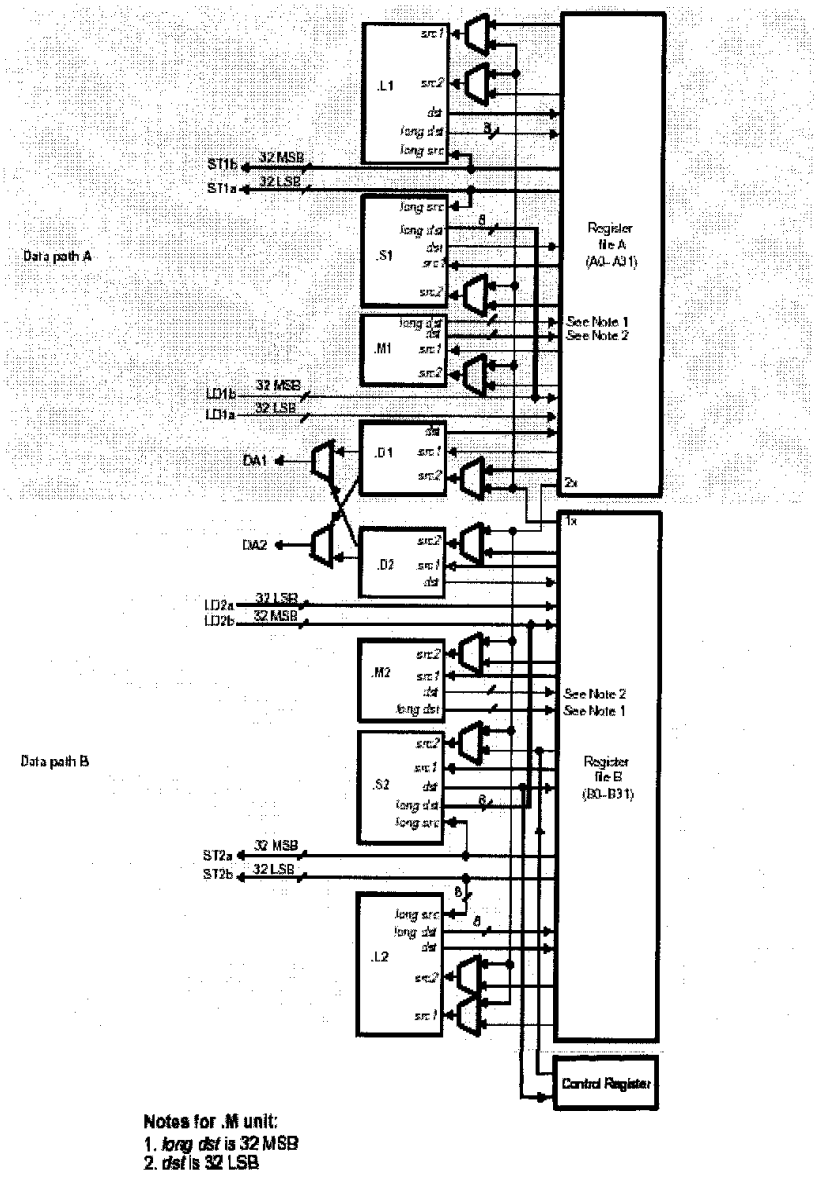


图 3.2 DM642 的数据通道图

3.2.2.2 DM642 的流水线

DM642 的硬件流水线包括取指令，译码，执行 3 级。其中，取指令阶段包括地址产生，地址发送，地址有效等待，指令包接受收；译码阶段包括指令分配和指令译码两个阶段；指令执行分为 5 个阶段，不同的指令执行所需要的阶段数不同。指令执行阶段如图 3.3

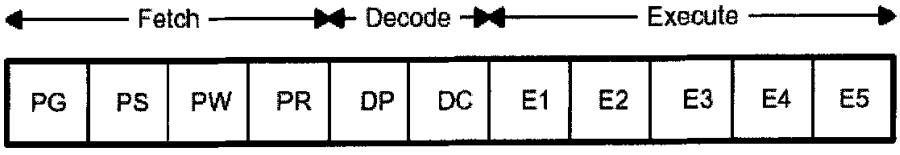


图 3.3 DM642 指令执行阶段图

当程序运行的过程中，如当指令包 n 的运行已经达到 E1 时，指令包 n+1 正在解码阶段，指令包 n+2 正在分配阶段，但这些指令包不同阶段在一个指令周期内同时执行。这样构成了软件的流水操作运行。如图 3.4

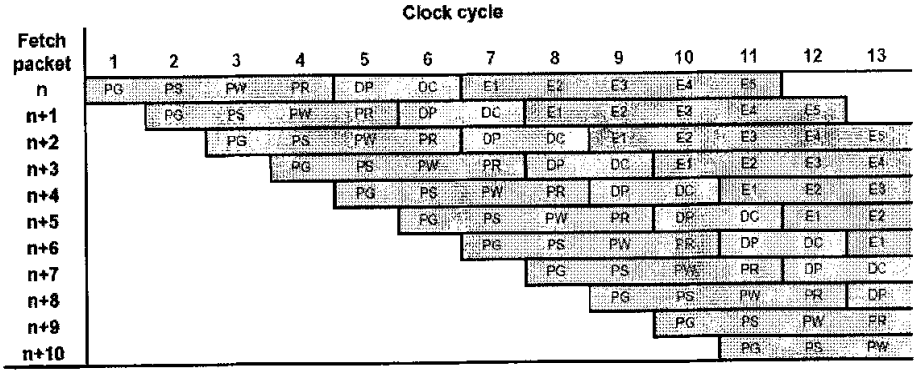


图 3.4 DM642 指令执行时序图

指令流水是按照机器时钟来执行的，也就是一个机器时钟周期的执行指令的一个阶段。在流水的过程中，当上一个指令包的目的操作数是下一个指令包的源操作数，而上一个指令包的目的操作数又不能在一个机器时钟内得到，则需要插入 NOP 指令，等待上一个指令包的操作数产生。

在 DM642 程序中流水越充分，也就是 NOP 指令越少，程序执行的越有效，所以优化的目的就是要在程序的流水过程中，应该注意那些影响流水线性能的地方尽量，减少 NOP

的数量。

一个取指包共有有 8 条指令组成，而一个取指包在执行前要拆分为最多 8 个执行指令包。每个执行指令包里的指令是并行的，而且每条指令在独立的处理单元中执行。每个数据指令包所拆分的执行指令包的个数是影响流水的重要因素，一种比较恶劣的情况就是 8 个指令中有 1 个指令需要和其他 7 个指令串行执行。另外一个影响流水的重要因素就是每个执行指令包里的指令的类型，因为每个指令执行的机器时钟数不同，则每个指令结束的时间不同，所以要在合适的地方插入 NOP 指令。最后还有一个影响流水的因素就是存储器系统的影响。所谓存储器系统对流水的影响主要体现在数据的读写指令中，在 DM642 中，有一个特点就是存储系统可以配置的，可以让程序存在一片地址空间中，让数据存在另外的一片存储空间中。程序的读取和数据的读取在流水线中是执行的同样的操作，只是在不同的阶段来完成而已。当存储器没有准备好响应 CPU 的读写时，就发生了存储器阻塞。如图 3.5，在 PW 阶段需要读出指令，在 E3 阶段需要读出数据，由于存储器的原因发生阻塞，从而使流水线的相应的阶段执行的时间超过一个时钟，需要另外的时钟时间来完成读写的操作。而不管流水是否发生阻塞，程序运行的结果是一定的。

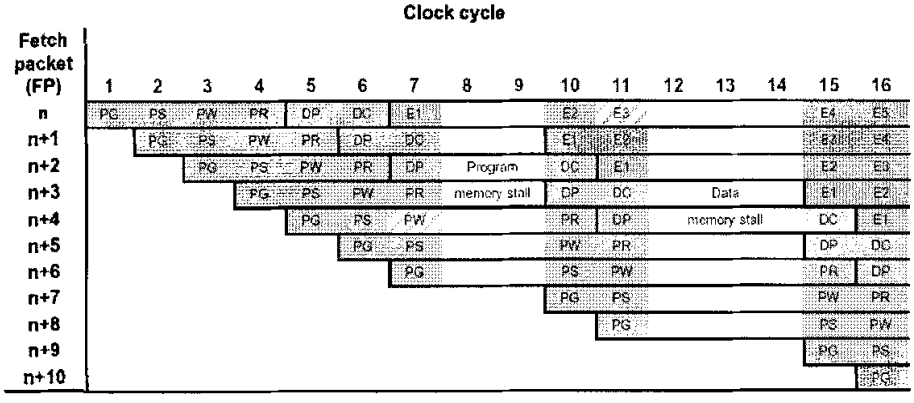


图 3.5 存储器阻塞造成的指令执行时序

3.2.2.3 DM642 的指令集

C6000 具有丰富的数据操作方面的指令，可以单指令实现比较大小，求绝对值，数据点乘等操作。DM642 的指令集在 C6000 的基础上，又补充了 8 位，16 位的扩充，非对齐的存取，数据的组合和拆分。指令表如图，每个功能单元都有自己可以执行的指令[32]。还可以

利用 A0，A1，A2，B0，B1，B2 作为条件操作寄存器。如图 3.6

.L unit	.M unit		.S unit		.D unit
ABS2	AVG2	SHFL	ADD2	SUB2	ADD2
ADD2	AVGU4	SMPY2	ADDKPC	SWAP2	ADDAD
ADD4	BITC4	SSHVL	AND	UNPKHU4	AND
AND	BITR	SSHVR	ANDN	UNPKLU4	ANDN
ANDN	DEAL	XPND2	BDEC	XOR	LDDW
MAX2	DOTP2	XPND4	BNOP		LDNDW
MAXU4	DOTPN2		BPOS		LDNW
MIN2	DOTPNRSU2		CMPEQ2		MVK
MINU4	DOTPNRUS2		CMPEQ4		OR
MVK	DOTPRSU2		CMPGT2		STDW
OR	DOTPRUS2		CMPGTU4		STNDW
PACK2	DOTPSU4		CMPLT2		STNW
PACKH2	DOTPUS4		CMPLTU4		SUB2
PACKH4	DOTPU4		MVK		XOR
PACKHL2	GMPY4		OR		
PACKL4	MPY2		PACK2		
PACKLH2	MPYHI		PACKH2		
SHLMB	MPYIH		PACKHL2		
SHRMB	MPYHIR		PACKLH2		
SUB2	MPYIHR		SADD2		
SUB4	MPYLI		SADDU4		
SUBABS4	MPYIL		SADDSU2		
SWAP2	MPYLIR		SADDUS2		
SWAP4	MPYILR		SHLMB		
UNPKHU4	MPYSU4		SHR2		
UNPKLU4	MPYUS4		SHRMB		
XOR	MPYU4		SHRU2		
	MVD		SPACK2		
	ROTL		SPACKU4		

图 3.6 DM642 的指令集

在 DM642 的程序运行时，优化的目的是尽量使各个处理单元能够并行执行。而实际情况中并行执行的限制的因素如下：

（1）处理单元的限制，即在同一个并行指令包中，任何两条指令不能在同一个处理单元中执行。如

ADD .S1 A0，A1，A2；

```
||  SHR .S1   A3, 15,  A2;
```

这两条指令同时使用S1处理单元，是非法的。

```
    ADD .S1   A0, A1, A2;
||  SHR .L1   A3, 15,  A2;
```

将两条指令分配给不同的处理单元执行，是合法的。

（2）交叉路径资源限制

每个指令执行包在同一个数据通道中执行的过程中，每次只能读取另一个数据通道的寄存器组的一个数据。例如S1处理单元的2个操作数能够从本数据通道的寄存器组A中读取，也可以一个数据从通过交叉路径读取，另外一个数据从本数据通道读取。而同一操作指令包的两条指令不能用同一个交叉通道。例如

```
ADD.L1X A0, B1, A1 ;
|| MPY.M1X A4, B4, A5 ;
```

两条指令同时使用交叉数据通道，所以是非法的。

```
ADD.L1X A0, B1, A1 ;
|| MPY.M2X B4, A4, B2 ;
```

两条指令分别使用不同的交叉数据通道，是合法的

另外DM642还扩充了一些特别的功能，同一个数据通道的处理单元可以在利用同一个交叉通道读取同一个数据，而且一定要读取同一个数据。如

```
ADD .L1X A0, B1, A1 ;
|| SUB .S1X A2, B1, A2 ;
```

两条指令同时使用交叉通道1X读取B1寄存器，是合法的。同时DM642还引入了交叉通道的阻塞，即当一条指令要通过交叉路径读取一个寄存器时，而这个寄存器的数据在先前的一个时钟周期的时候更新过，则需要在这条指令执行前插入一个时钟的延迟，也就是自动插入一个NOP指令。如

```
ADD .S1, A0, A0, A1;
ADD .S2X A1, B0, B1;
```

在第二个ADD指令通过交叉通道读取A1之前，由于第一个ADD指令对寄存器A1的数据进行了更新，所以在第二个ADD执行之前要插入一个时钟待更新完成。

（3）在DM642中，寻址通道DA1和DA2都和两个D处理单元联系在一起的，在存取数据的时候，可以用一个数据通道的寄存器寻址来读数据，而用另外一个数据通道的寄存器寻址去

写数据。但是，两个使用同一个源地址或者目的地址的存取指令不能同时执行，D处理的寻址寄存器只能在同一边上。

```
LDNW  .D2T2 *B2[B12], B13;
|| LDB   .D1T1 *A2, A14;
```

两条指令的源地址的寄存器不是和LDNW的D2T2在同一边，所以是非法的。

```
LDNW .D2T2 *B2[B12], A13;
|| ADD .D1x A12, B13,  A14;
```

这两条指令符合上面的要求，是合法的。

（4）两个指令不能同时写同一个寄存器。

（5）同一个周期内不能读同一个寄存器超过4次。

3.2.3 DM642 的外围电路

DM642 提供了强大的外设，包括可以配置的片上内存，以及无缝连接的 SDRAM 外存接口，强大的数据搬移引擎 EDMA，方便的视频输入输出接口，以及网络接口等如图 3.7。

3.2.3.1 DM642 的 CACHE 结构

为了解决 CPU 和存储器之间的速度匹配问题，解决由于低速存储器和高速的 CPU 之间的矛盾。DSP 提供了两级缓存来解决这个矛盾，一级缓存的速度最快，CPU 可以全速访问，但容量较小，二级缓存容量较大，但是速度较慢，只有 CPU 速度的一半。DM642 采用改进的哈佛结构，一级缓存包括数据缓存和代码缓存各 16Kb，这样在程序运行时，指令的操作码和操作数可以在同一个周期里取到。DM642 的二级缓存有 256Kb，可以编程配置为一部分为可寻址的片上 RAM，一部分为 CACHE，或者全部为可寻址片上 RAM，或者全部为 CACHE。DM642 的缓存中的区间还可以编程配置为映射特定的外部 SDRAM 空间[11]。

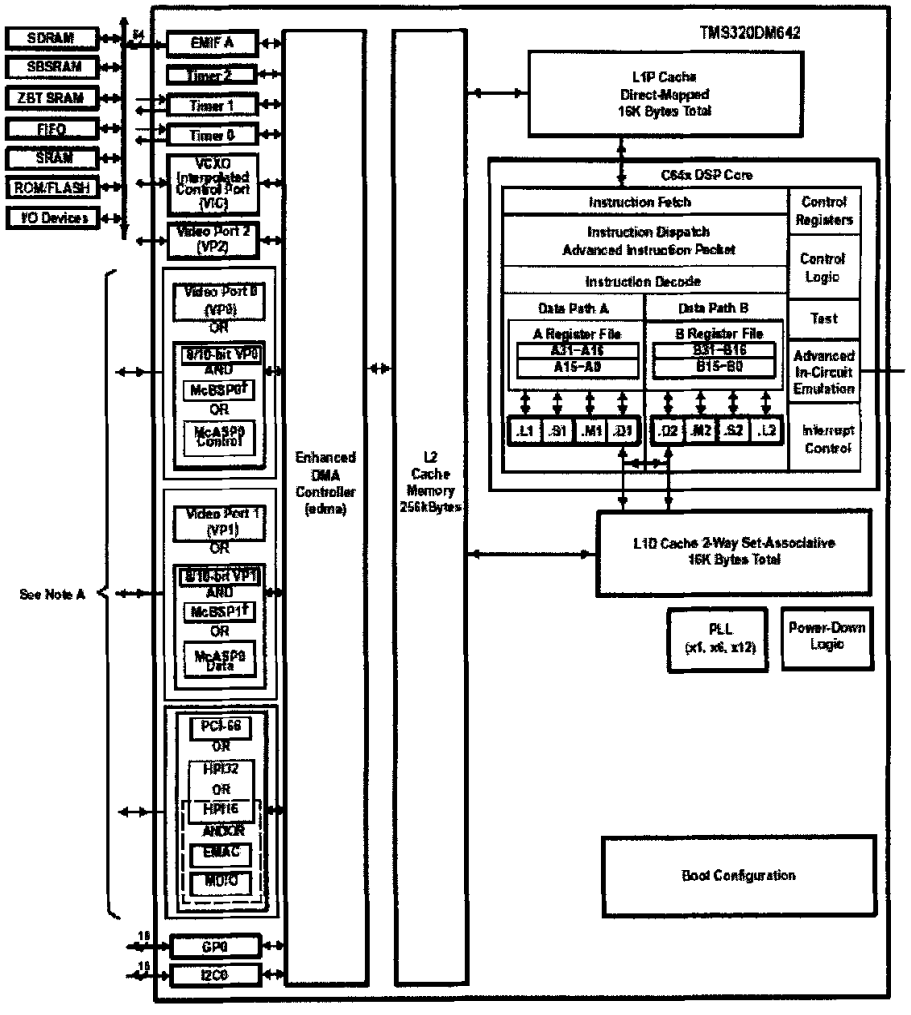


图 3.7DM642 外围接口图

3.2.3.2 DM642 的视频接口

DM642 提供了 3 个视频端口，可以灵活的配置成视频采集或者视频回放模式。在图像采集情况下，每个视频口可以采集 2 路 8/10 位的视频图像；在回放模式情况下，每个视频口可以输出一路 8/10/16/20 视频流。每个视频端口都配置有专门的数据通道，数据通道包括 2560B 的 FIFO 和 EDMA 通道组成。在视频端口工作时，每个 FIFO 可以作为一个大的缓存使用，也可以分为 1280B，640B，640B 大小 3 个缓存来缓冲 Y，Cb，Cr 分量。而且有 EDMA 来负责数据的搬移，不用增加 CPU 的负荷，就可以把数据采集到合适的地方，或者把视频

数据回放出去[23][24]。

3.2.3.3 DM642 的以太网口

DM642 提供了 10/100M 网络接口。网络接口提供 EMAC 接口模块和物理层数据输入输出管理模块[25]。EMAC 模块控制 DSP 和物理层之间数据的传输，数据输入输出模块负责物理层的配置和状态监控。

3.2.3.4 DM642 的其他接口

DM642 还提供其他很丰富的接口，如存储扩展接口，主控制器端口，多路缓存串口，通用 I/O 端口等。存储扩展端口可以与 SDRAM，SRAM，ROM 等无缝连接，同步存储器的时钟可以外接，最高频率可以达到 133M，最高可以扩展 256M 的存储空间。主控制端口可以让主机直接访问 DSP 的内存空间，包括存储空间和映射了地址空间的外设的寄存器，这样可以不用增加 DSP 开销的情况下，完成数据的通信。多路缓冲串口是一种同步串口，可以提供全双工的串行通信，可以与工业标准的编解码器，其他串行 A/D，D/A 连接。DM642 最多还可以提供 16 个通用 I/O 口，可以编程实现电平和边沿事件，这些时间可以作为中断源或者触发 EDMA 的事件[3]。

3.3 本章小结

本章简单介绍了各种经典的多媒体处理 DSP 器件的特点。重点介绍了以下 DM642 的 CPU 核的体系结构，指令执行流程和 DM642 的指令集，这些都是多媒体处理库优化的基础，优化的目标就是要充分利用资源，尽量不让那些降低系统的性能的情况出现。最后简单介绍了 DM642 的外设情况。

第四章 基于DM642的MML视频编码库优化

4.1 优化平台

MML 视频库开发的软件平台是 TI 公司开发的软件开发套件 CCS。CCS 是 TI 公司为该公司的 DSP 软件开发而提供的软件工具，它集成了 DSP 系统开发过程中的各个步骤所需要的工具。而且 CCS 还提供了一种简单的基于 DSP 的操作系统 DSP/BIOS，在 DSP/BIOS 的基础上，还可以通过在不停止处理器运行的情况下观察程序运行的每个阶段的数据，及时发现程序或者是硬件的问题。同时，CCS 还可以通过图像来观察 DSP 的数据，使得项目开发中可以更加直观的分析程序的效果，这个功能在图像处理和视频应用的工程中特别重要。CCS 还是一个开放的集成开发环境，第三方可以根据自己的需要添加模块，使得 CCS 具有更加灵活的应用。

CCS 是 DSP 软件开发的重要工具，也是一个很方便的 DSP 项目开发工具。它设置方便，支持多个 CPU，可以同时配置多个硬件系统。软件方面可以同时管理多个工程，对于源文件，头文件，库文件分开管理，可以很容易的识别。CCS 还可以通过图形界面来配置实时功能，中断向量的定义，存储器的映射，线程的定义[26][27][28]。

CCS 集成编译环境可以支持标准 C 语言，汇编语言以及两种语言的混合编译。CCS 具有相当高的代码编译技术，具有高效的 VLIW 生成工具，对于 C 代码编译优化器的优化的效率可以达到 70—80%；对于线性汇编代码优化编译器的优化效率可以达到 95—100%，而经过手工优化汇编代码的效率可以达到 100%。CCS 高效的 C 编译效率和图形化的选项功能大大节约了开发者的时间。

CCS 的调试功能支持 C 和汇编的混合调试，具有专门观察变量的窗口。可以同时对多个 CPU 进行调试，对于相同的 DSP 可以同时对多个板卡进行调试。在程序运行的过程中，可以将数字信号直接存入文件中，还可以对代码段的性能进行测试，分析。另外 CCS 还提供了虚拟的数据环境，可以将同一组数据用不同的格式表现出来，例如对图像数据可观看图像的时域图，频域图，FFT 变换图，眼图等等。CCS 还提供了多线程应用程序分析工具，在工程应用中对线程级别的代码进行分析[30][31]。

4.2 结构优化

考虑到 DSP 嵌入式系统的特点，要对基于 PC 或者其他的 CPU 的编码器结构流程进行调整，来适应 DSP 的体系结构，提高编码器的运行效率，提高编码速度。PC 上的程序往往只重视功能的实现，程序代码结构和数据的安排只是从实现的难易上考虑，而 DSP 嵌入式程序受到 DSP 硬件资源的限制，对程序结构和数据流的组织上都有很大的影响，包括代码和数据的存储地址，而且一些特别为 PC 上运行而写的代码根本不适合在 DSP 上执行。实际上，在 DSP 系统上，对代码的大小具有很强的敏感性，有时候在选择复杂的算法在 PC 上可以提高编码的效果，而在 DSP 系统中，也许会由于代码量的大小，反而会降低 DSP 编码器的效果。所以 DSP 的程序要尽量符合 DSP 的存储结构和缓存结构[8][9]。对代码结构的优化，主要是减少不必要的代码的运行，以及充分利用 DSP 的指令集特点，根据代码和数据的相关性，安排代码的处理流程。

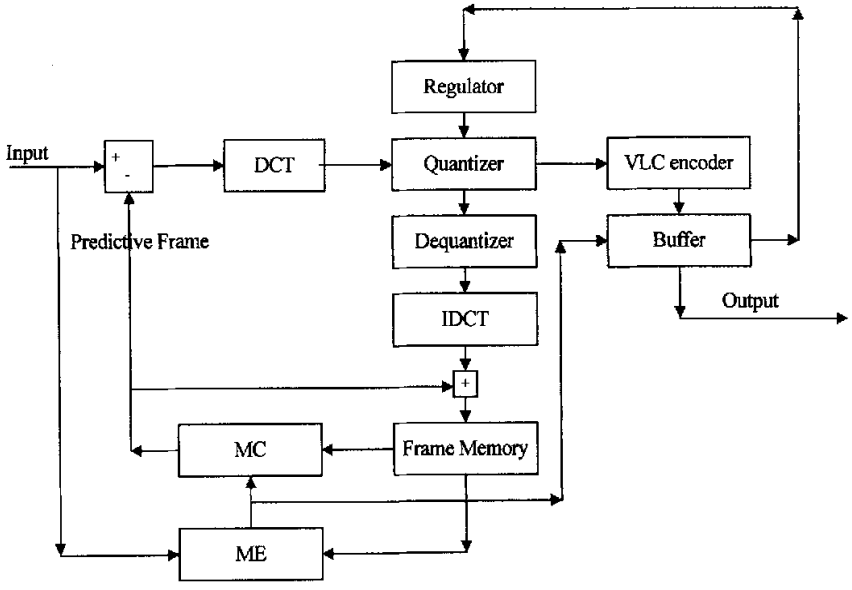


图 4.1 视频编码结构框图

在视频编码应用中，无论是 H263，H264，还是 MPEG2，MPEG4 这些标准都是采用的都是相同的编码框架，即基于 DCT 和差分编码的混合编码结构。该编码器由 DCT 变换、量化、反量化、反 DCT 变换、运动估计与补偿和变长编码（VLC）几个部分组成。其中 DCT 变换、量化、反量化、反 DCT 变换、运动估计与补偿构成了个反馈环。这种编码结构中采

用的运动估计是在空间域中进行的，称为空间域运动估计。算法的主要使用 DCT 和量化来减少视频信号的空间冗余，运动估计来减少视频的时间冗余，然后采用熵编码提高编码效率。这些编码标准的结构框图如图 4.1。

对于 MPEG-4 的视频编码主要分为 I 帧编码和 P 帧编码，I 帧编码全部是帧内编码，整帧的图像编码是在一个大的循环中执行的，P 帧编码包括帧内编码和帧间编码，编码过程中，宏块的编码模式在帧内模式和帧间模式之间互相切换。

对于 I 帧编码，不需要其他的帧图像来作为参考，只是在帧内图像的变换压缩。在初始化编码器的参数后，I 帧编码只需要在输入原始的图像数据，经过一系列的变换，量化，编码就可以得到 I 帧图像的码流，而经过反馈回路的反量化，反变换等一系列的操作重建该图像，作 P 帧的编码参考帧之用。I 帧主要以宏块为单位形成一个大的循环，当编码图像的大小发生改变的时候，只需要改变循环次数就可以了，不用改动程序的具体结构。循环的核心主要前向通道和反馈通道两部分组成。前向通道输入原始图像，由变换量化函数，帧内预测函数，CBP 编码，宏块的熵编码，输出 I 帧图像的码流。反馈通道输入熵编码前的图像数据，经过帧内预测恢复，反量化，反 DCT 变换，得到一个和原图像类似的参考图像。其中，把变换和量化封装在一个函数中，减少了局部的缓存的大小，然后把 ZIGZAG 扫描和熵编码全部放在同一个函数中。同时在 ZIGZAG 扫描和熵编码前，通过判断 CBP 是否编码，来决定是否要对本宏块进行编码。I 帧编码主要有原始数据存储空间，变换量化系数存储空间，以及码流缓存空间。其中，原始数据在 DCT 变换后和帧内量化后存储的是同一个缓存空间，节约了缓存的空间。循环中的程序流程图如图 4.2。

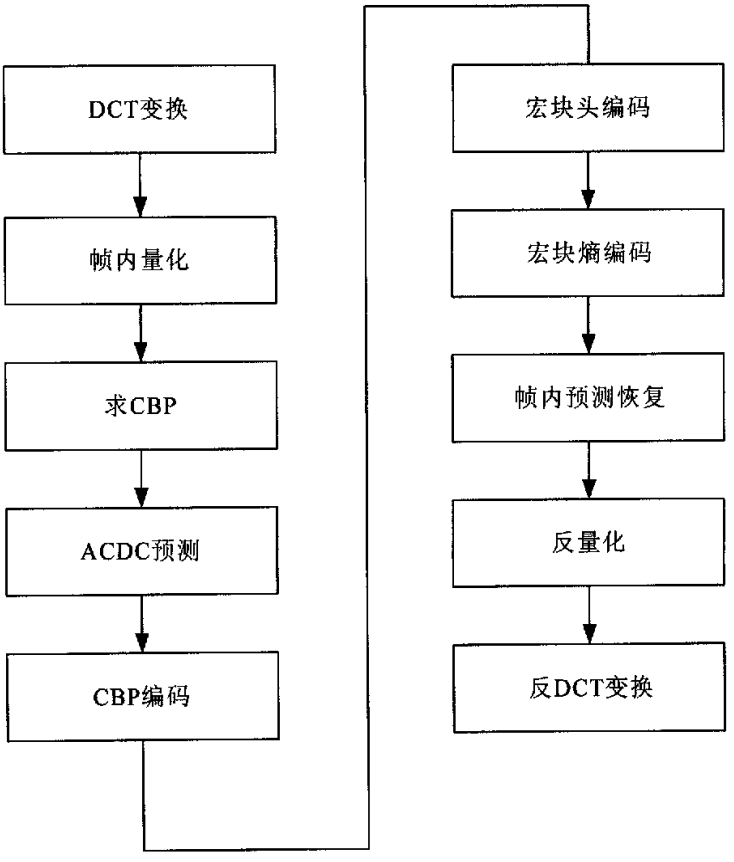


图 4.2 MPEG4 I 帧图像编码流程图

对于 P 帧编码，编码模式包括帧间编码和帧内模式两种编码模式。在初始化编码器参数后，P 帧编码器需要输入当前帧图像和参考帧图像，经过运动估计，经过一系列的变换，量化，编码就可以得到 P 帧图像的码流，而经过反馈回路的反量化，反变换等一系列的操作重建该图像，作下一个 P 帧的编码参考帧之用。P 帧也是主要以宏块为单位形成一个大的循环。循环的核心主要运动估计，前向通道和反馈通道两部分组成。运动估计部分输入原始图像和参考帧图像，搜索到合适的参考宏块，输出运动向量和残差；前向通道由变换量化函数，CBP 编码，宏块的熵编码，输出 P 帧图像的码流； 反馈通道输入熵编码前的图像数据，经过运动补偿，反量化，反 DCT 变换，得到一个和原图像类似的参考图像。其中，把变换和量化封装在一个函数中，减少了局部的缓存的大小，然后把 ZIGZAG 扫描和熵编码全部放在同一个函数中。同时在 ZIGZAG 扫描和熵编码前，通过判断 CBP 是否编码，来决定是否要对本宏块进行编码。P 帧编码主要有原始数据存储空间，参考窗的数据存储空间，变换

量化系数存储空间，以及码流缓存空间。其中，原始数据在 DCT 变换后和帧间量化后存储的是同一个缓存空间，节约了缓存的空间。循环中的程序流程图如图 4.3。

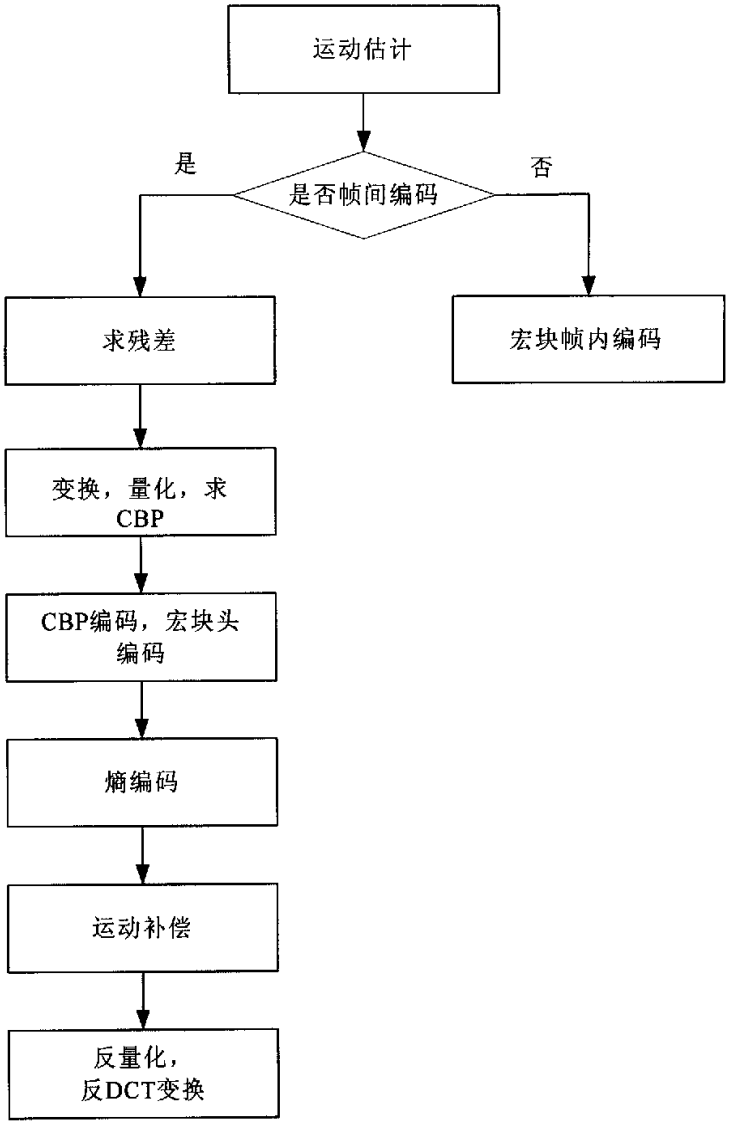


图 4.3 MPEG4 P 帧视频编码流程图

对于 H.264 视频编码也是基于 DCT 和 DPCM 的混合结构编码，增加了很多的先进的编码技术，主要包括支持 1/4 或 1/8 像素精度的运动矢量，4×4 块的整数变换，提供了标准的 UVLC 和 CABAC 熵编码，先进的帧内预测模式，这些技术使得 H.264 的编码效率有了很大程度的提高。但是鉴于更高级的帧内编码的技术应用，P 帧中采用帧内编码模式的宏块

的比例也比 MPEG-4 的 P 帧高，所以将 IPP 的编码流程将 I 帧编码和 P 帧编码的循环都放在了一个循环里面，对每一个宏块的编码模式都进行一次判断，待编码模式确定后，每个宏块中分别以 4x4 的块为单位进行编码。

在 IPP H.264 的实现过程中，将 I 帧和 P 帧都用同一个循环来实现如图 4.4。循环主要包括 3 个部分，宏块编码模式选择，前向通道，后向通道。其中，宏块编码模式选择输入的条件有是否为帧内 SLICE 和当前帧的图像，P 帧图像输入还包括其参考帧图像。输出该宏块的编模式。

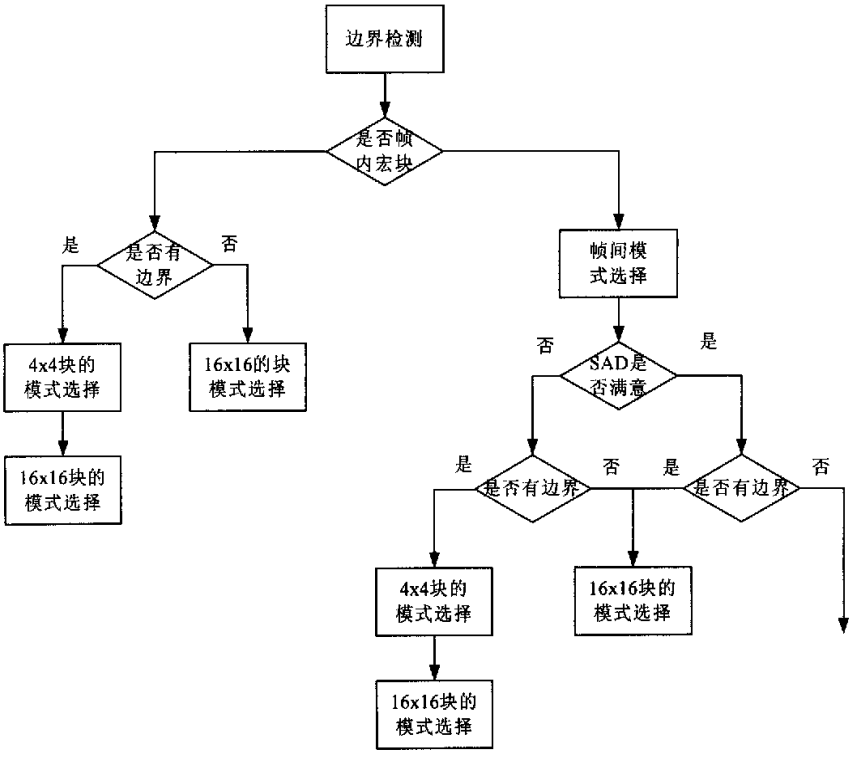


图 4.4 H.264 视频编码流程图

每个宏块在选择编码模式的算法都不大相同，主要用宏块内是否有边界以及宏块是否在帧内 SLICE 里。在判断宏块是否有边界的算法如下：对于任何一个待编码的宏块，分别求出水平方向和垂直方向每个相邻的两行之间的差值的绝对值之和，当水平方向或者垂直方向的绝对值之和大于阀值，则认为这个宏块里存在边界。然后根据是否有边界和是否帧内 SLICE 来决定宏块编码模式选择的算法。当当前宏块属于帧内宏块时，如果该宏块存在边界时，则同时进行 4x4 块上和 16x16 块上的进行帧内预测方式的遍历以决定预测的模式；如果

该宏块不存在边界时，只是在 16x16 块上的编码方式的遍历以决定预测的模式。当当前宏块是帧间宏块时，首先对帧间的编码模式进行遍历，得到帧间编码模式最优的 SAD 值，当本宏块有边界存在且帧间编码的 SAD 值大于阀值时，且本宏块有边界存在的时候，就要进行帧内的 4x4 块上和 16x16 块上的预测模式的遍历以决定具体的预测方式是帧内预测还是帧间预测，当判断本宏块有边界和帧间预测的 SAD 大于阀值两个条件只有一个符合时，只是进行 16x16 块上进行预测模式的遍历来决定预测模式，当帧间的 SAD 小于阀值时，直接选择在帧间编码的模式。

而编码的前向通道和后向通道基本上和 MPEG-4 的前向和后向相同。主要的不同在于 H.264 的前向通道和后向通道中，都以 4x4 的块为单位进行变换，量化。下图为 H.264 的流程图。

4.3 存储器优化

4.3.1 DM642 的存储体系和缓存机制

从 DSP 应用的观点来看，最理想的情况是有一个大容量高速度的片上内存，可是在过去的几十年里，处理器的速度越来越快，虽然存储器的速度也比以前的速度提高了很多，但是相对于处理器的速度增幅来说，存储器的速度还是远远不够的，所以出现了一个 CPU 和存储器之间速度的矛盾。虽然出现了可以匹配 CPU 的高速存储器，但是那需要占用很大的体积，而且成本很高。

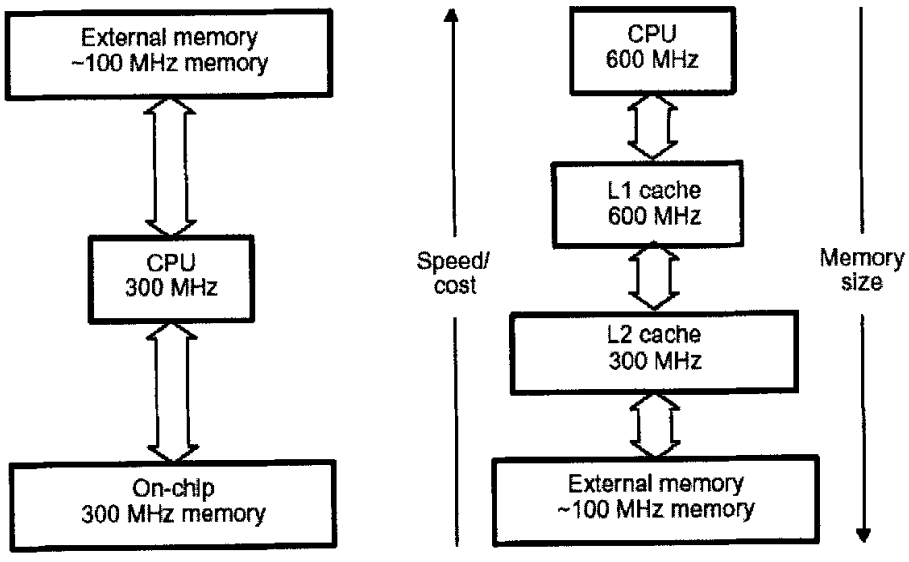


图 4.5 DSP 存储体系结构图

在平行的存储结构中，如图 4.5，如果 CPU 和片上存储器的速度都是在 300MHz，则当 CPU 读写片上存储器时，不会有时钟阻塞，但是当 CPU 读写外存的时候，就会每次会产生 2 个时钟周期的阻塞。如果 CPU 工作在 600MHz 的时候，每 2 个时钟周期才能对片上存储器进行一次读写，也就需要阻塞一个时钟。当程序运行在循环中时，CPU 反复的读写片上内存，阻塞的频率会更高，CPU 总是隔一个周期阻塞一个周期，也就相当于 CPU 运行在 300MHz 的时钟下。然而，由于存储器技术发展的速度总是跟不上处理器发展的速度，在片上造一块体积小，速度和处理器速度一致的存储器的成本是相当高的。

为了解决这种问题，引入一种分级存储体系如图。把容量小，速度快的存储器作为 CPU 的最高一级的存储器，CPU 访问它不需要时钟阻塞，低层的存储器采用容量较大，速度较慢的存储器。这样形成一种离 CPU 越近，速度越快的分级存储体系结构。最典型的结构就是将高层的存储器作为缓存，有专门的缓存管理单元来控制。通过这种方式，CPU 访问存储器的频率大约就是最快的那一级的存储器的速度，而避免了访问低速存储器所需要的大量时间。

DM642 的存储系统由片上内存和片外内存两部分组成。其中片上内存采用两级缓存体系，第一级缓存包括互相独立的 L1D 和 L1P 缓存，第二级缓存是片上 SRAM（L2）。鉴于

DM642 的哈佛体系结构，L1D 和 L1P 分别为高速的代码和数据缓存。L2 是一个统一的程序/数据空间，可以作为一个二级缓存，也可以作为一个可寻址的 SRAM，还可以把 L2 的一部分配置成缓存器，另一部分配置成片上可寻址的 SRAM，SRAM/CACHE 混合模式。而片外的存储系统一般有 SDRAM 构成，构成大容量第三层的存储器。

图 4.6 是 DM642 的存储体系结构图，主要有 L1，L2，SDRAM 以及负责数据传输的 EDMA。

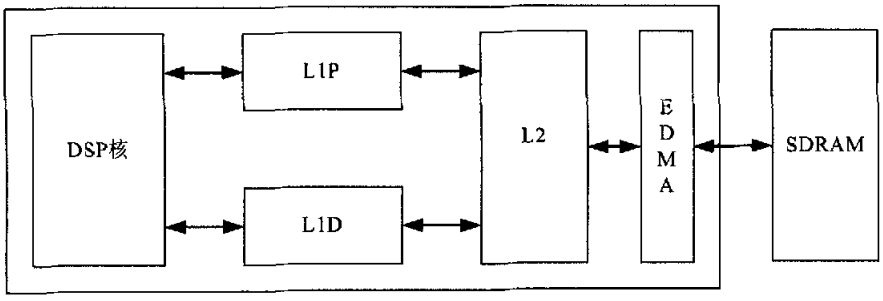


图 4.6 DM642 存储体系结构图

L1P 是一个直接映射缓存，每行宽度为 32BYTE。所谓直映射模式就是每行缓存映射特定在地址的数据。每个可被缓存的存储空间只是映射到缓存的一个地址中，所以在缓存控制器只需要检测一个标签 RAM 就知道数据是否在缓存中。一般，DSP 的算法总是包含很多循环，也就是 DSP 对于一些代码核总是执行很多次。L1P 的大小达到 16KBYTE，足够存储 DSP 算法中反复执行的循环核的大小。当 CPU 在 L1P 中没有命中的时候，L1P 就向 L2 缓存取一行代码。因为程序具有相当高的相关性，所以当读取第一条指令后，后面的指令就不需要等待直接可以缓存得到。

L1D 是 2 路关联缓存，每行宽度 32BYTE。2 路关联是比直接映射更加灵活的方案，这种缓存模式对数据的缓存更有随机行，有更大的步长，更加适合于 DSP 数据的缓存。在 2 路关联缓存结构中，每路缓存对应 8KBYTE 的数据，在 2 路关联缓存的结构中，每个内存空间地址在一路缓存中都对应一行缓存。这样，每个数据在 L1D 中就对应两行缓存，在更新数据的时候，缓存控制器采用 LRU 算法。和 L1P 一样，由于数据的相关性，在缓存第一个数据以后，后面的数据就可以直接从缓存得到，不需等待。而且由于数据缓存是双口的，可以让 DSP 核同时访问两个数据。

L2 是介于 CPU 和外部存储器之间的存储模块，可以配置为 5 中工作方式。L2 分为

32KBYTE，32KBYTE，64KBYTE，128KBYTE 大小的存储器四部分，可以如图 4.7 配置为一部分缓存，一部分为片上 SRAM，或者全部配置成缓存器，或者全部配置成片上 SRAM。L2 存储器可以看作是 CPU 和外部可寻址存储空间的主要缓存，特别是当外部存储空间和内部一级缓存的容量相差很大的时候，L2 缓存更始必不可少的。L2 缓存的工作方式与 L1D 类似，采用 4 路关联缓存模式。

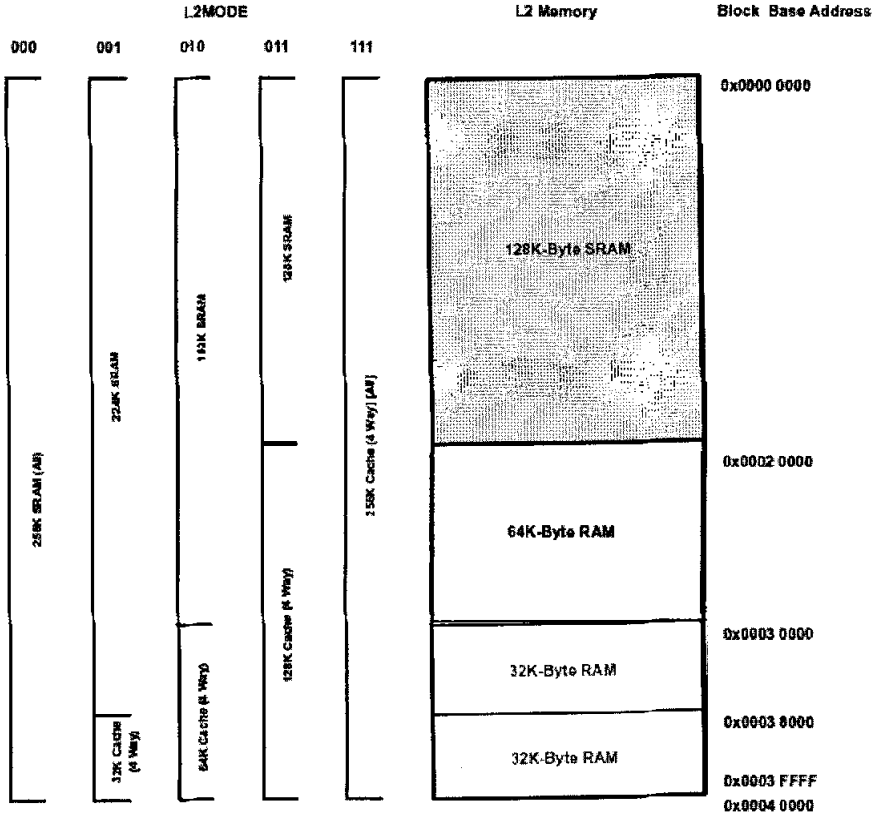


图 4.7 DM642 中 L2 配置图

最外层存储结构主要由大容量 SDRAM 组成，主要由 EDMA 负责 L2 和 SDRAM 之间的数据通信。

DM642 的整个缓存体系的工作过程如图 4.8：当 CPU 读数据的时候，首先检查 L1 缓存的标签 RAM，如果命中则直接读入，如果未命中，则向 L2 发送读入指令，如命中则将数据读入 L1，在送给 CPU，否则发送读指令给 EDMA，由 EDMA 将数据读入 L2，在送入 L1 直到 CPU。在 CPU 写出数据时，如果在 L1 中命中时，则将数据写入 L1，并将对应的标签设置为脏字，随后回写到 L2，直到 SDRAM。如果未在 L1 中命中，由于 L1D 只能为读数

据开辟缓存空间，CPU 将数据直接回写到 L2 缓存中，随后将数据回写到 SDRAM 中。

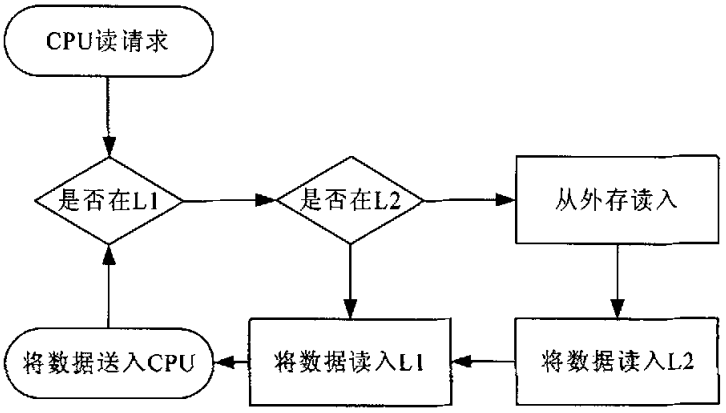


图 4.8 DSP 缓存机制图

4.3.2 缓存优化和存储器分配

在 DSP 的系统应用中，最理想的情况当然是平行结构的存储体系，而且有一个足够大，速度与 DSP 核速度一致的 RAM，这样 DSP 就不会出现时钟阻塞。而实际上，随着 DSP 的速度越来越快，分级缓存的引入，CPU 总是会出现时钟阻塞。因此在分级缓存体系中，CPU 运行的频率越高，程序运行的速度也就越快，这就需要尽可能减少时钟阻塞的出现。在系统应用中，采用适合缓存体系的算法会大大提高系统的速度。

缓存在系统优化中的作用主要体现在缓存行的重用上，当 CPU 访问的数据不在缓存中时，就发生阻塞，等待数据所在的缓存行读入缓存中来。当该缓存行读入到缓存中，以后 CPU 访问该行的数据时，就不用再需要 CPU 阻塞。因此利用 CACHE 来优化的目标就是为了让缓存行的重用最大化。

DM642 CPU 发生阻塞的情况主要有以下几种：

（1） 交叉通道阻塞：当指令要读取一个先前一个时钟更新的寄存器时，CPU 阻塞一个时钟。

（2） 当指令同时读入同一个 L1D 块的时钟时，CPU 阻塞一个时钟。

（3） 读 L1D 未命中，CPU 阻塞 6 或者更多的时钟。

（4） 写 LID 时，回写缓存满，则 CPU 阻塞一个时钟。

（5） 读L1P未命中，CPU阻塞8或者更多的时钟。

由以上分析可知，缓存优化的关键就是提高L1的效率，L1的效率提高了自然意味着L2的效率也提高了。可以从两个方面提高缓存的效率。

第一个方面是提高L1的命中率：通过将相关的数据放在同一个缓存行中来增大缓存行的重用次数；通过让CPU访问的数据路数不超过缓存的路数来避免那些还在重用的缓存行被剔除；让CPU分开访问缓存行来延长缓存行在缓存中的时间；采用LRU剔除策略来剔除那些不需要的缓存行。第二个方面是尽量减少每次未命中时CPU等待的时间：尽量让CPU的访问请求在L1未命中时，在L2中可以命中。

在应用层面上，优化主要有以下几个方面：

（1） 选择合适的L2的配置方式：合适的数据对于CACHE的优化是相当关键的。由于DM642的数据缓存是4路关联的，在缓存的大小上要注重考虑，如果CACHE开的过大，则会造成浪费，而且会使很多关键数据不能永久驻留片上，从而增加了CPU阻塞的可能，缓存开的过小，则会导致很多关键的数据和代码不能全部驻留，从而使部分核心数据和代码总是换进换出，也会增加CPU阻塞的发生。

（2） 合理布置代码和数据的存储地址：由于缓存的工作机制是建立在代码和数据的高度相关性上的。所以在数据和代码的存储安排上，可以将核心代码和关联性很强的数据尽量存在相邻的地址上，这样可以在缓存过程中全部换入CACHE中。在大的循环核中，可以将循环核的数据和代码放在相邻的地址上，从而可以全部放入CACHE中。

由于DM642的L2容量为256K，根据经验和实验得到，在视频应用中，由于存在大量的数据。需要在L2上开辟大量的空间来存储当前编码的宏块，所以一般所开的CACHE大小为64K，剩下的全部配置为可寻址空间。64K的CACHE足够把视频处理的大部分核心代码同时包含。而剩下的192K的可寻址片上SRAM，可以存储很多重要的数据和代码，这样可以让他们永远驻留在片上，增加了L2的命中率。在视频编码的处理过程中，主要存储空间的分配如表（以CIF图像格式为例）。而存储在L2的代码包括量化，变换，编码等核心代码，而那些板级初始化等只需要执行一次的函数就放在SDRAM中如表4.1。

数据或者代码块	用途	大小（字节数）	存放位置
Image_cur	当前帧图像	99K	SDRAM
Image_ref	参考帧图像	99K	SDRAM
Image_refh	参考帧水平插值图像	99K	SDRAM
Image_refv	参考帧垂直插值图像	99K	SDRAM
Image_refhv	参考帧斜线插值图像	99K	SDRAM
Dct_codes	DCT 变换系数	384	SRAM
qcoeff	量化后的系数	384	SRAM
Block_cur	当前宏块	384x2 (ping-pong 结构)	SRAM
Block_ref	参考窗	384x18 (ping-pong 结构)	SRAM
Block_refh	参考窗水平插值	384x18 (ping-pong 结构)	SRAM
Block_refv	参考窗垂直插值	384x18 (ping-pong 结构)	SRAM
Block_refhv	参考窗斜线插值	384x18 (ping-pong 结构)	SRAM

表 4.1 MPEG4 编码器存储分配表

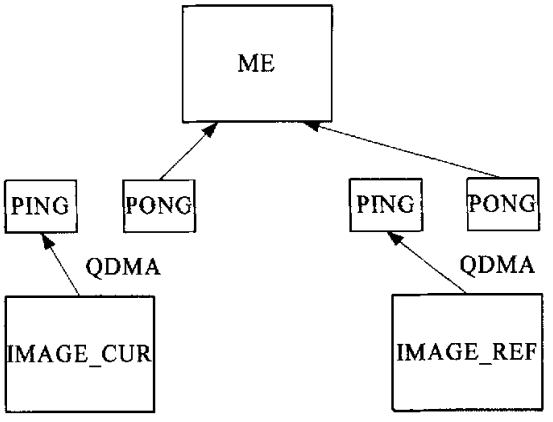
4.3.3 EDMA 数据流组织

DM642 的 EDMA 模块是一个高效的数据转移模块，在每个 EDMA 时钟可以转移 8 个 BYTE 的数据。当 CPU 的数据为 600MHz 时，EDMA 的带宽可达 2.4GB，EDMA 负责 L2 与外部存储器以及外设之间数据的搬移。数据的搬移可以由 CPU，外部事件等等各种条件触发。在 EDMA 搬运数据的过程中，首先由 CPU，或者外部事件来提出搬运请求，搬运请求要求包括源地址，目的地址，搬运性质。所有的搬运请求根据优先级的不同排成队列，组成 Q0，Q1，Q2，Q3 四个搬运请求队列。EDMA 控制器将优先集最高的请求参数送入 EDMA

控制寄存器，执行最终的数据搬移。

DM642 的 EDMA 提供了一种 CPU 同步的数据搬移请求 QDMA。在视频应用中，数据的搬移主要由算法的数据流驱动，而不是外界设备驱动的。所以在视频应用中，QDMA 的数据搬移一般是帧同步的，而为了充分利用 QDMA 的资源，可以将 EDMA 分配到几个不同的优先集队列中[1]。

由于视频应用算法是以数据流驱动的，数据流的组织就显得相当重要。DM642 的数据流主要有 EDMA 负责搬移，DSP 核负责数据的处理。两者配合工作，可以大大提高 DSP 的速度。如果二者配合不好，就会出现 DSP 的持续等待，从而会大大降低处理速度。本文提出了 PING-PONG 机制如图 4.9，来调度 EDMA 和 CPU 核之间的工作，具体的工作过程为：首先为 EDMA 的目的地址开辟 PING-PONG 两个缓存，当 EDMA 搬移数据入 PING 缓存的同时，可以让编码引擎从 PONG 缓存中取数据进行处理。这样 EDMA 和 CPU 核可以同时运行，提高了两者的并行性。EDMA 数据流组织的优化的目标是尽可能提高 EDMA 的并行度，使 CPU 可以充分运行。以编码中的 ME 函数为例：



如图 4.9 PING-PONG 机制体系结构

在 ME 从两个 PONG 缓存中得到原始数据，得到当前宏块和参考窗的数据进行运动估计。在 CPU 进行运动估计的过程中，QDMA 则把下一个宏块的数据和参考窗的数据传入 PING 数据，达到 CPU 和 EDMA 的同时执行。

4.4 代码优化

4.4.1 代码优化流程

传统的 DSP 软件优化流程，在 PC 上验证 C 代码的正确性，然后将 C 代码手工翻译为汇编语言。这种方法既耗时间，又消耗大量的人力，而且容易出错，最后为系统的维护和升级也带来了很大的难度。

TI 公司针对 DSP 软件开发的难度，提供了一系列的代码开发的辅助工具。对于一些痕次要的函数，可以用这些代码开发的辅助工具来完成优化工作，还可以这些工具完成指令选择，并行安排，流水这些烦琐的工作。可以把开发工作者从汇编语言的编写中解脱出来，专心面向市场竟可能快的开发出相应的产品。同时，由于代码的大部分还是 C 代码，也利于后来的产品维护和升级。

利用这些代码开发工具开发的主要过程有 3 个阶段。第一个阶段包括 C 代码的验证，然后通过分析决定那些部分占用的机器周期最多，也就是后来优化的主要对象。第二个阶段主要是把尽可能多的已知条件告诉编译器，使编译器更加有效的工作。主要途径包括尽量减少潜在的指针对齐问题；允许循环的结尾有不可知的操作数；利用伪代码告诉编译器循环的次数；伪代码进行存储单元的对齐；利用特定的多媒体指令代替部分 C 代码,；扩展数据的带宽等等方法。第三个阶段主要是对一些核心代码进行优化，主要用一些线性汇编，甚至手工排流水来完成代码的优化[50][51]。

代码优化的流程具体如图 4.10，在第一阶段，可以在不需要任何 C6000 背景知识的情况下，完成代码的部分优化，汇编程序的生成。如果代码的表现不如意，则进入第二个阶段，用以上的系列方法，改变编译器的优化选项和伪代码来告诉编译器优化的方向和重点，为了得到更好的优化结果，只能够将核心代码改写为线性汇编代码，甚至改写为汇编代码以提高整个函数运行的效率。

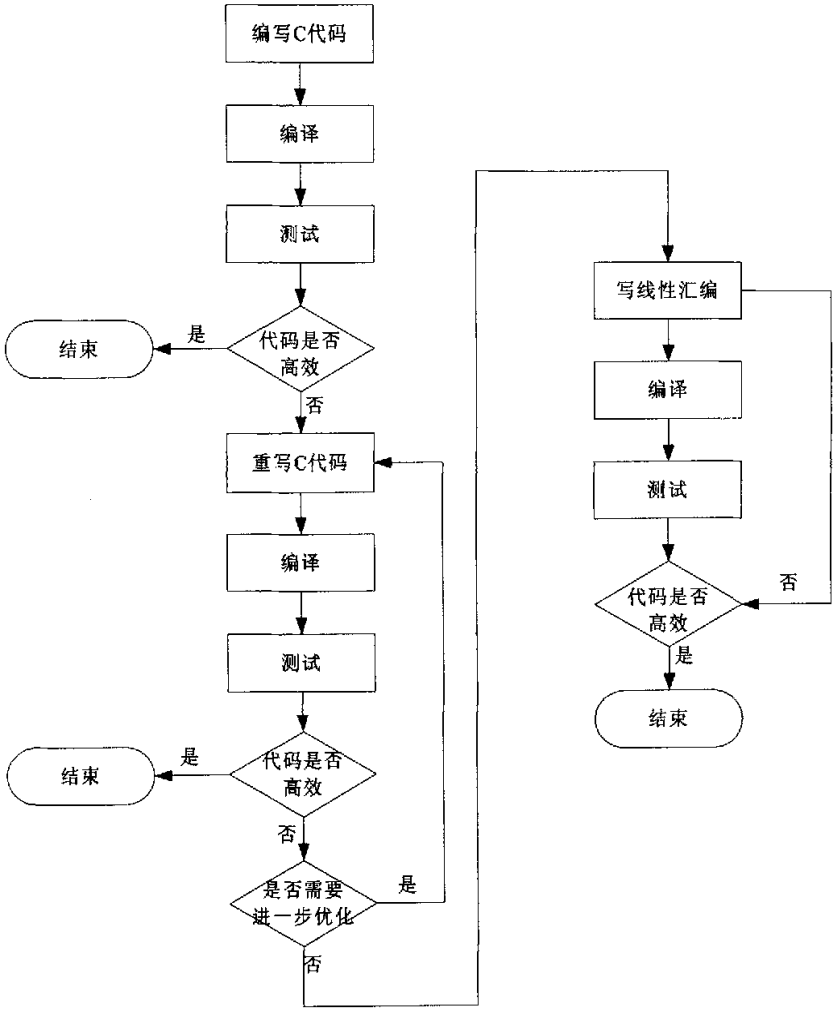


图 4.10 优化步骤流程图

4.4.2 代码优化方法

4.4.2.1 CCS 优化选项

TI 的编译开发工具 CCS 提供了一系列优化选项，针对某些不同的标准对程序进行优化，程序开发者可以根据自己关心的方面选择具体的选项，得到想要的结果[29]。

基本优化选项

-g：全符号调试，此选项可以防止过度优化导致的代码量大大膨胀而造成的代码执行缓慢。

No –ms，ms0，ms1，ms2，ms3：代码量和代码速度优化选项，no –ms 表示不用考虑代码量，只考虑代码的性能；ms0，ms1，ms2，ms3 选项随着数字的变大，代码量作为优化的标准的比重越来越大，即 ms0 首先考虑代码的性能，其次考虑代码量，ms1 也是先考虑代码的性能，但代码量在优化的评价上占的比重增大；ms2 首先考虑代码量，再考虑代码的性能；ms3 同样先考虑代码量，但代码性能在代码性能优化评价的比重变小。

-o1，-o2，-o3：代码的优化级别选项，随着数字的增加，代码优化的程度就越高，但是代码量也会变大。随着优化的级别增加，一些工具也会应用到优化中去，例如软件流水技术，单指令多数据技术等。

-pm -op0，-pm -op1，-pm -op2，-pm -op3：项目级别优化选项，-pm –op0 表示函数或者变量会被外部的代码引用；-pm –op1 表示函数不会被外部的代码引用；-pm –op2 表示变量不会被外部的代码引用；-pm –op3 表示函数和变量都不会被外部的代码引用。

高级优化选项

-ol0，ol1；RTS（run-time support function）选项

-oi ：内联函数选项，在代码大小小于设定的界限时，函数被自动定义为内联函数。

-me ：大小端选项

-ml0，ml1，ml2，ml3 ：表示数据和函数的访问调用方式。访问数据和函数有以下两种方式：near，far

对于数据：near 是指数据访问通过 DP（指向.bss 起始段位置的指针）来访问，只需一条 load 指令，但所访问的数据距 DP 的最大偏移量不能超过 32K；在 far 的情况下，则要用 MVKL，MVKH，load 三条指令来访问数据，没有 32K 的限制。

对于函数调用：near 指函数跳转是相对跳转，即 label 跳转，只需要一条指令，但最大只有 2M 的偏移量；far 则是指函数跳转是绝对跳转，根据寄存器的内容跳转，能有 32 比特的偏移量。

其他优化选项

-mt ：消除存储器关联选项

-mu ：关闭软件流水，用在逐步调试中。

-mw ：使编译器产生附加的反馈信息，供用户分析代码的性能。

-k ：保留产生的汇编文件，用来观察优化的结果。

4.4.2.2 扩展读写带宽

在 C 代码中，很多循环的都是以象素为单位进行循环的，所以循环的次数很大。而由于 C6000 的存储带宽是 32bit，而每次只是取到一个象素，只能够串行执行；而且在循环体中，load 和 store 指令所占用的周期数很大，load 取到的数据是后面指令操作的目标，所以也不能够并行执行，这样 C6000 的资源并没有充分的应用。在优化的过程中，可以改写循环体里的代码，每次循环操作 4 个甚至 8 个象素。这样的改写有如下几个优点：

每次 load，store 指令存取的数据更多了，减少了 load，store 执行的次数。

每次循环体里处理的有 4 个或者 8 个象素，减小了数据的相关性，可以提高指令的并行性。

在视频处理库中，以块的求均值为例 C 的源代码如下

```
void MML_AverageBlock_MPEG4_8u( const unsigned char*   pSrc1,
                                const unsigned char*   pSrc2,
                                unsigned char*   pDst
                                )
{
        int i=0;
        for(i = 0; i < 64; i++)
        {
            pDst[i] = (unsigned char)((pSrc1[i] + pSrc2[i] + 1) >> 1);
        }
}
```

以上代码经过，增加每次循环处理的象素个数为 4 个，代码如下

```
void MML_AverageBlock_MPEG4_8u1( const unsigned char*   pSrc1,
                                 const unsigned char*   pSrc2,
                                 unsigned char*   pDst
                                )
{
```

```
        int i=0;
        for(i = 0; i < 16; i++)
        {
            pDst[i] = (unsigned char)((pSrc1[i] + pSrc2[i] + 1) >> 1);
            pDst[i*4+1] = (unsigned char)((pSrc1[i*4+1] + pSrc2[i*4+1] + 1) >> 1);
            pDst[i*4+2] = (unsigned char)((pSrc1[i*4+2] + pSrc2[i*4+2] + 1) >> 1);
            pDst[i*4+3] = (unsigned char)((pSrc1[i*4+3] + pSrc2[i*4+3] + 1) >> 1);
        }
}
```

每次循环处理的象素个数为 8 个时，代码如下

```
void MML_AverageBlock_MPEG4_8u2( const unsigned char*   pSrc1,
                                 const unsigned char*   pSrc2,
                                 unsigned char*   pDst
                                )
{
        int i=0;
        for(i = 0; i < 8; i++)
        {
            pDst[i] = (unsigned char)((pSrc1[i] + pSrc2[i] + 1) >> 1);
            pDst[i*8+1] = (unsigned char)((pSrc1[i*8+1] + pSrc2[i*8+1] + 1) >> 1);
            pDst[i*8+2] = (unsigned char)((pSrc1[i*8+2] + pSrc2[i*8+2] + 1) >> 1);
            pDst[i*8+3] = (unsigned char)((pSrc1[i*8+3] + pSrc2[i*8+3] + 1) >> 1);
            pDst[i*8+4] = (unsigned char)((pSrc1[i*8+4] + pSrc2[i*8+4] + 1) >> 1);
            pDst[i*8+5] = (unsigned char)((pSrc1[i*8+5] + pSrc2[i*8+5] + 1) >> 1);
            pDst[i*8+6] = (unsigned char)((pSrc1[i*8+6] + pSrc2[i*8+6] + 1) >> 1);
            pDst[i*8+7] = (unsigned char)((pSrc1[i*8+7] + pSrc2[i*8+7] + 1) >> 1);
        }
}
```

在优化级别选项为 o2 时，即不加入软件流水的情况下函数执行的需要时钟值如表 4.2

函数名	每次循环一个象素	每次循环四个象素	每次循环八个象素
执行需要的时钟个数	1167	831	757

表 4.2 宏块求均值优化对比表

有上表可知，在每次循环体中有四个象素的时候，代码优化有了很大的提高，函数消耗的时钟个数大约提高 30%，而每次循环的象素为 8 个时，优化结果提高不大。

4.4.2.3 减少判断，跳转语句

代码中，频繁的出现判断语句或者跳转的语句。对于 C6000 这种能够并行运行的体系结构的 CPU 来说，是最不愿意看到的。原因如下：第一 C6000 有 8 个处理单元，需要代码编译后能够大量的并行运行，而频繁的出现判断语句或跳转语句，不能够实现并行处理，而且跳转指令需要插入很多 NOP 指令等待跳转完成，大大降低了代码的性能；第二 在循环体中，如果出现判断和跳转语句，会打破循环的流水，造成执行的时钟数成倍的增加。

对于优化判断和跳转语句多的代码主要有调整语句减少 if-else 语句，尽量改为逻辑判断；尽量把判断和跳转语句放到循环体的外面；把跳转的代码可以换成查表的方法来完成。

在 C 代码中有大量的 if-else 语句，而执行的操作有很少的时候，可以尽量用逻辑判断来减少 if-else 语句，还可以把 if-else 的语句改写为相应的线性汇编来减少 if-else 语句。因为 C6000 体系中提供了 5-6 个判断寄存器，在代码执行的过程中，可以先判断这些寄存器的条件是否符合而决定该行代码是否执行，而且寄存器的判断和该行代码的执行在同一个时钟里完成，可以大大提高代码的执行速度。例如在反量化函数中，最后要进行饱和处理，代码如下：

```
static unsigned short saturate(unsigned level)
{
 if (level > 2047)
 {
    level=2047;
```

```
    }
    else if (level <-2048)
    {
        level = -2048;
    }
    else
        level =(unsigned short)level;
    return (level);
}

static inline unsigned short L_shl(unsigned a)
{
    return ((unsigned short)(_shr((_sshl((a),(20))),(20))));
}
```

把判断和跳转语句改在循环外面执行，是很有效的方法，也是C代码优化的常用方法。由于起、其方法比较简单，这里就不再详细讨论。

最后一种方法是用查表发来代替判断语句也是一个有用的方法，在熵编码可以得到广泛的应用。例如在VLC编码的过程中，主要根据run-level查表实现VLC的编码。在查表的过程中，要通过判断这个level是否是最后一个非0的系数，是否是帧内编码来决定查哪个表。实际上，可以通过造一个四维的表coeff_VLC[intra][last][abs_level][run]来减少判断的次数，首先定义两个变量intra，last，其中intra代表是否为帧内系数，last表示是否为最后一个系数。在实际的编码过程中，不用判断具体的条件，可以直接用intra，last作为四维数组的下标去寻址，可以查到合适的码字。

4.4.2.4 使用内联函数

DM642的指令集提供了很多多媒体专用指令，而在C语言中，这些指令的优势是很难体现出来的，而且编译器也不能够很好的应用这些多媒体专用指令，无法体现DM642的优势。所以TI推出了大量的内联函数，这些内联函数与那些特有的多媒体指令直接影射，通

过这种办法把多媒体的专用指令嵌入到 C 代码中，发挥 DM642 的优势。如数据的打包指令，在 C 中需要多次移位和逻辑操作才能完成，而利用专用的 PACKL4 指令可以很容易的把存在 4 字的低 8 位组成 1 个 32 位的双字。

半象素搜索也是 MPEG-4，H264 标准中的重要组成部分，半象素搜索的前提就是要插值生成半象素图像。而在实际的半象素插值包括水平方向的插值，垂直方向的插值，斜线方向的插值。如图 4.11

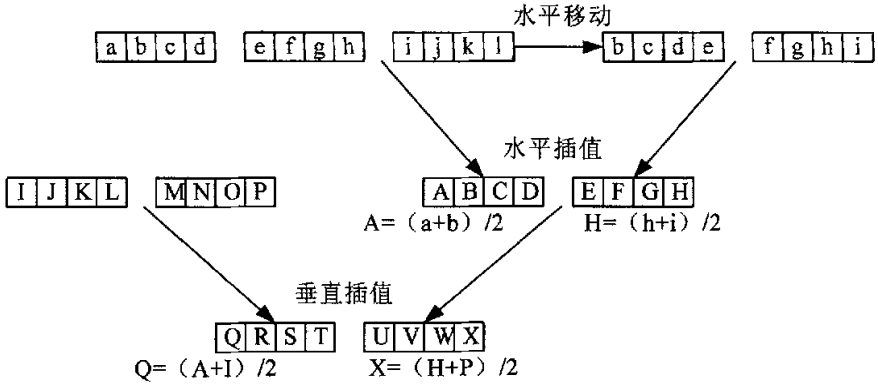


图 4.11 插值流程图

在水平插值的过程中，由于每个象素值以字节为单位存储的，一个双字节存储了 4 个象素值。在水平插值过程中，需要一个象素值和右边的象素值求均值，这个就可以利用_shlmb 函数来可以构造偏移一个象素值的双字。再引用_avgu4 直接对每个象素求均值，大大降低了函数的复杂度。

4.4.2.5 TI 图像处理库

图像，视频处理中有一些常见的数据处理操作，如 DCT 变换，IDCT 变换，SAD 计算等。TI C6000 系列 DSP 作为专门为多媒体应用而开发的 DSP，TI 公司专门提供了视频，图像处理常用的函数。这些函数已经经过优化，运行效率非常高，所消耗的时间最短。由于还提供了源码，我们可以借鉴这些函数的方法，用到 MML 库函数中。

虽然 TI 函数的库函数运行效果很高，但是在象素的存储上安排比较特别，传入的参数

要特别的处理，所以运用时要特别小心。例如对于 DCT 变换的过程，要求 64 个象素以 ZIGZAG 的顺序排列，而实际上由于在系统中数据的象素值总是和象素的物理位置有关的，无形中为了运用函数库增加了代码开销。

4.2.4.6 线性汇编

线性汇编语言是 TMS320C6000 中独有的一种编程语言，介于高级语言和低级语言之间。与 C6000 汇编代码相比，不同的是编写线性汇编代码不需要指明使用的寄存器，指令的并行与否，指令的延迟周期和指令使用的功能单元，汇编优化器会根据情况确定这些信息。这样可以减少编程的工作量，缩短开发周期。线性汇编的基本格式与汇编语言相同，它也是 ASCII 文件，扩展名为特定的.sa，线性汇编文件使用一些汇编优化器伪指令来区分线性汇编和通常的汇编代码，.cproc 命令和.endproc 命令限定了汇编优化器优化的代码段，.cproc 命令放在代码段的开始，.endproc 命令放在代码段的结尾；.reg 命令使汇编优化器为数值，选择一个寄存器，这个寄存器与对该值进行操作的指令所选择的功能单元一致，还可以定义一个寄存器对来存储代码中大于 32bit 的数据；.trip 命令指出循环的迭代次数。一条线性汇编语句与汇编语句相似，可以包括 6 个部分：标号，条件，指令，处理单元，操作数和注释。运算指令不一定要指定，可以由汇编器自行安排。如果程序员安排了处理单元，实际上是命令汇编器按照特定的处理单元来完成，有时反而限制了汇编器的优化，降低了优化的效果。但是，在优化的过程中，如果程序员对线性汇编有足够的了解，自己为指令安排处理单元可以得到很好的优化效果，总之，如果程序员有信心在指定了处理单元后，优化的效果比以前好的，就可以自己安排处理单元。以下以函数为例：

在 MPEG-4 的量化中，由于在量化的过程中常常要出现除法，而在 DM642 中是不存在除法指令的。在 DM642 平台上实现一次除法需要大量的操作才能完成，所以在实际的线性汇编中，用所有的 QP 的倒数造一个倒数表，由于 QP 的倒数都是小数不适合定点 DSP 运算，再将每个 QP 的倒数左移一定的位数，形成最后的 QP 倒数表。这样在量化的过程中，只需要量化前的参数乘以 QP 的倒数，右移一定的位数就可以了，大大降低了运算的复杂程度。用乘法和移位来代替除法，有一个问题就是引入了运算误差，但对 PSNR 基本没什么影响。DM642 提供了 MPY2 指令，MPY2 指令可以一次完成 2 对 16 位数的乘法，即两个 32 位数用 MPY2 相乘时，低 16 位与低 16 位相乘，高 16 位与高 16 位相乘，结果存在寄存器对中。

函数部分代码如下：

```
    for (i = 0; i < 64; i++)
{
        short acLevel = pSrcDst[i];

        if (acLevel < 0)
        {
            acLevel = (-acLevel) - quant_d_2;
            if (acLevel < quant_m_2)
            {
                pSrcDst[i] = 0;
                continue;
            }
            acLevel = (acLevel * mult) >> SCALEBITS_H263;
            sum += acLevel;
            pSrcDst[i] = -acLevel;
        }
      else
      {
            acLevel -= quant_d_2;
            if (acLevel < quant_m_2)
            {
                pSrcDst[i] = 0;
                continue;
            }
            acLevel = (acLevel * mult) >> SCALEBITS_H263;
            sum += acLevel;
            pSrcDst[i] = acLevel;
        }
    }
```

优化后主要的指令如下，在编写线性汇编的过程中，关键因素除了，减少循环次数，增大每次循环执行象素的个数外，选择合适的指令也是一个重要的优化途径，在选择线性汇编指令的过程中，要尽量把指令分布在不同的处理单元中，使每个处理单元的符合大致一样。优化后结果如表 4.3

	优化前（周期数）	优化后（周期数）	速度提高（%）
8x8 块的量化	3354	219	93

表 4.3 块量化优化对比表

```
LDNDW *src_address++,src1:src0
LDNDW *src_address++,src3:src2

SHR2 src0,15,sign_flag0
SHR2 src1,15,sign_flag1
SHR2 src2,15,sign_flag2
SHR2 src3,15,sign_flag3

SADDUS2 sign_flag0,v_1,sign_flag0
SADDUS2 sign_flag1,v_1,sign_flag1
SADDUS2 sign_flag2,v_1,sign_flag2
SADDUS2 sign_flag3,v_1,sign_flag3

MPY2 sign_flag0,quant_d_2,quant_add1:quant_add0
MPY2 sign_flag1,quant_d_2,quant_add3:quant_add2
MPY2 sign_flag2,quant_d_2,quant_add5:quant_add4
MPY2 sign_flag3,quant_d_2,quant_add7:quant_add6

PACK2 quant_add1,quant_add0,quant_add0
PACK2 quant_add3,quant_add2,quant_add2
PACK2 quant_add5,quant_add4,quant_add4
```

```
PACK2 quant_add7,quant_add6,quant_add6

SUB2 src0,quant_add0,src0
SUB2 src1,quant_add2,src1
SUB2 src2,quant_add4,src2
SUB2 src3,quant_add6,src3

ABS2 src0,src0
ABS2 src1,src1
ABS2 src2,src2
ABS2 src3,src3

CMPGT2 src0,quant_m_2,dst0
CMPGT2 src1,quant_m_2,dst2
CMPGT2 src2,quant_m_2,dst4
CMPGT2 src3,quant_m_2,dst6

ADD dst0,nonzero,nonzero
ADD dst2,nonzero,nonzero
ADD dst4,nonzero,nonzero
ADD dst6,nonzero,nonzero

XPND2   dst0,dst0
XPND2   dst2,dst2
XPND2   dst4,dst4
XPND2   dst6,dst6

AND   src0,dst0,src0
AND   src1,dst2,src1
AND   src2,dst4,src2
```

```
AND   src3,dst6,src3

MPY2 mult,src0,dst1:dst0
MPY2 mult,src1,dst3:dst2
MPY2 mult,src2,dst5:dst4
MPY2 mult,src3,dst7:dst6

PACKH2 dst1,dst0,src0
PACKH2 dst3,dst2,src1
PACKH2 dst5,dst4,src2
PACKH2 dst7,dst6,src3

MPY2 src0,sign_flag0,dst1:dst0
MPY2 src1,sign_flag1,dst3:dst2
MPY2 src2,sign_flag2,dst5:dst4
MPY2 src3,sign_flag3,dst7:dst6

DOTP2 src0,v_1,src0
DOTP2 src1,v_1,src1
DOTP2 src2,v_1,src2
DOTP2 src3,v_1,src3

ADD src0,A_sum,A_sum
ADD src1,A_sum,A_sum
ADD src2,B_sum,B_sum
ADD src3,B_sum,B_sum

PACK2 dst1,dst0,dst0
PACK2 dst3,dst2,dst2
PACK2 dst5,dst4,dst4
```

```
PACK2 dst7,dst6,dst6

MV dst2,dst1
MV dst6,dst5

STNDW dst1:dst0,*dst_address++
STNDW dst5:dst4,*dst_address++ +
```

4.4.2.7 软件流水

软件流水是对循环过程的一种重构，使循环过程可实现流水执行或部分流水化，其目的在于改进循环过程的执行并行性。这种重构是由编译器支持的，因此称之为软件流水重构的可能性是基于：循环的各次迭代之间没有相关，这时可以从不同次迭代之间抽取指令，重构迭代过程，从而使得循环的迭代过程中各语句能够全部或部分(更好地)流水化执行，由此获得更高的指令级并行性。循环重构方法：选取原来循环过程的不同迭代过程中不相关的语句，组成新的迭代过程，从而重构循环。例如一个循环体中，每次的循环包括指令包括 A，B，C，D，E 五条指令，而且这五条指令不重用相同的处理单元，所以可以构成如图 4.12 的软件流水。

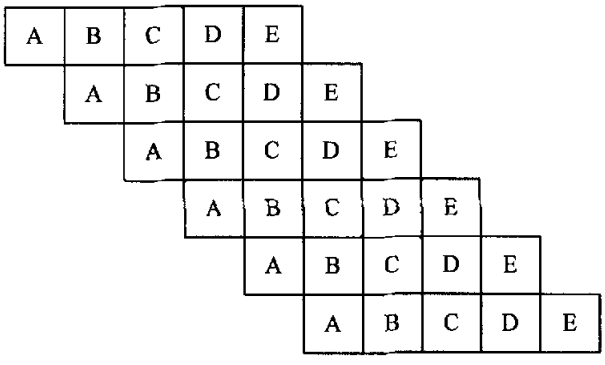


图 4.13 软件流水示意图

根据上图可知，在没有软件流水的情况下，循环的次数为 n 次需要执行的周期数为 5n，而在软件流水的情况下只需 n+8 个时钟，大大减少了循环运行的时钟个数。软件流水虽然

能够很大的减少循环执行的时钟个数，但是在应用过程中，对代码要求也是比较严格，所以在应用软件流水时要特别注意。

由于在嵌套循环中编译器仅对最里面的循环执行软件流水，因此对于执行周期很少的内循环进行循环展开，对外循环进行软件流水，这样可以改进 C 代码的性能。循环展开的次数越多，跳转就会越少，这对于减少流水线中的阻塞是很有好处的，但是展开次数的增加不可避免的带来了代码长度的增加，这也会影响到程序执行的速度。比较好的做法是对循环展开次数作为参数进行实验，测量不同展开次数下的程序的执行周期，进行量化分析，这样就可以在进行编译优化的时候选择适当的循环展开次数。

尽管软件流水循环可以包含内联函数，但是不能包含函数调用。在软件流水过程中绝对不能出现函数跳转的，有两种方法可以解决这个问题，对于循环里面的函数可以定义为内联函数，这样在编译的过程中，就可以把函数展开，从而去掉函数的跳转指令，另外一种方法就是分拆循环，把函数移到函数外面。

另外在循环中不可以有条件终止指令；在循环体中不可以修改循环控制变量；如果循环体内复杂的条件代码需要超过 5 个条件寄存器或者代码尺寸需要 32 个寄存器以上，则这个循环不可以进行软件流水。对于循环中有条件终止指令的，需要在条件终止指令前，插入无效操作指令以便把条件终止指令移到函数体外。如果循环体内复杂度过大，需要超过 5 个条件寄存器和 32 个通用寄存器时，可以将大的循环体分拆为几个小的循环体，分为几个小的软件流水来实现。

在 CCS 中。只要在编译时，使用-o2 选项和-o3 选项，编译器可对循环代码实现软件流口。但在具体的编译过程中，也许有某些条件不符合，而不能形成流水，这是就要改写 C 代码或者线性汇编代码，使他符合软件流水对代码的要求。在使用 CCS 汇编器产生软件流水的过程，需要不断的尝试，改写代码直到产生的汇编代码满意为止。

4.4.2.8 采用指令乱序技术

程序中，有些指令的执行顺序没有严格的要求，则可以通过作出一些位置上的调整，穿插于其他的指令之中，从而减少指令的相关性，增加运行的并行性。尤其在循环里，当循环体较小的时候，可以把多个循环的代码写在一个循环体里，合并成一个循环，从而减小循环内指令的相关性，增加指令运行的并行性，但是要注意不要使循环过于复杂，以至不能进行软件流水线的优化。

4.5 部分代码优化结果比较

在插值类的函数中，优化注重在增加每次循环处理象素的个数，每次循环同时对四个象素进行操作。将四个象素值都存储在一个寄存器中，每次对这个寄存器操作就是对四个象素值同时进行操作。同时对四个象素进行操作，对于指令的选择也是优化的一个重要方式，在插值过程中，几个关键的指令包括 SHLMB，AVGU4 等等，另外还要采用软件流水的技术。表 4.4 是采用这些技术以后，插值类函数优化前后的结果对比。

函数名	优化前的函数周期	优化后的函数周期
MML_Copyblockhalfpel	268/946/245/628	104/144/207/244
MML_CopyMBhalfpel	851/3612/617/3913	275/361/429/362
MML_Reconblockhalfpel	413/385/916/711	223/186/278/301
MML_Sad8x8	91/218/199/293	70/99/156/184

表 4.4 插值类函数优化前后的效果对比

注：有四个值是表示整像素点，垂直半像素插值，水平半像素插值，斜线拌像素插值的情况。

在量化类函数中，由于量化的数据需要扩展取值范围，每个寄存器中只能保存两个待量化的象素值，每次指令操作只能操作两个象素的值。然而为了增加操作指令和操作数的非相关性，采用指令乱序技术，在循环体中每个指令都重复执行一次，这样可以处理 4 个象素。由于量化中还存在一些判断语句，可以采用 SSHL，SHR 指令来减少饱和操作中的判断语句。表 4.5 是量化类函数优化前后的结果对比。

函数名	优化前的函数周期	优化后的函数周期
MML_Quantinvintra_h263	2055	355
MML_Quantinv_h263	2382	345
MML_Quant_h263	3872	294
MML_Quantintra_h263	2743	215

表 4.5 量化类函数优化前后的效果对比

在求均值，求差值，求和值类函数中，最关键的就是提高每次循环处理的象素的个数，但是当每次的循环处理的象素个数过大，则造成循环次数过小，降低软件流水的效率降低，

在块的求和类函数中，每个循环的操作四个象素与每次循环操作八个象素的执行效率相当。这类函数的关键是将四个象素值存储在同一个寄存器中，同时选用 SADDU4，SUBABS4 指令作为主要指令。求均值，求和，求差三类函数大体相同，以求均值为例，表 4.6 是求均值优化前后对比。

采用上述方法，以下是部分编码模块的优化效果对比：

函数名	优化前的函数周期	优化后的函数周期
MML_Averageblock	684	91
MML_AverageMB	2761	235

表 4.6 求均值优化前后效果对比

利用前面优化后的函数库来搭建一个 MPEG-4 的编码器（图像大小为 CIF 格式），再用几种典型的测试序列对该编码器进行了测试，表 4.7 列出了优化前后的时间信息（取的都是前 10 帧）、帧率。

序列	优化前	优化后
Basketball_qcif	5.82	165.44
News_qcif	7.36	186.12

表 4.7 MPEG4 编码器优化前后对比图

4.6 本章小结

本章首先简单介绍了视频处理库的优化软件平台 CCS。然后介绍了一下 MPEG4 以及 H.264 编码器结构调整，这种调整是为了适合 DSP 的体系结构，提高编码器中图像数据的复用程度，增加存储空间的利用效率。存储器的优化主要介绍了 DM642 的存储体系和缓存机制，以及在 MPEG4 编码器中具体的代码和数据存储空间分配等问题，在存储器优化方面还介绍了 EDMA 数据流组织在编码器中的应用方法。在低层函数的优化方面主要关注代码的优化，而很少考虑存储器和数据流方面的问题，主要介绍了一些具体的优化方法，包括扩展读写带宽；减少判断，跳转语句；使用内联函数；线性汇编优化；软件流水以及指令乱序等技术。

第五章 总结和展望

多媒体技术特别是视频压缩是当今国际上研究的热点，而且已经得到了广泛的应用。DSP 作为多媒体技术实现的重要平台，也成为人们热衷的研究对象。

本文首先对多媒体技术领域作了简要的介绍，视频编解码标准，音频编解码标准，以及图像处理的一些标准和技术是多媒体库的主要组成部分，也是多媒体处理库函数实现的理论基础。而多媒体技术实现的物理基础主要分为基于 PC 实现，基于 DSP 实现，基于 ASIC 实现，而在嵌入式多媒体应用系统中几乎主要基于通用 DSP 实现的。

其次，本文对多媒体库的主要函数进行了分析，特别是对多媒体函数进行分层封装。用户可以根据对多媒体的了解来调用相应的函数，当然调用高层函数比较方便，但缺少了系统的灵活行。

然后，本文又介绍了各种经典的多媒体处理 DSP 器件的特点。重点介绍了以下 DM642 的 CPU 核的体系结构，指令执行流程和 DM642 的指令集，这些都是多媒体处理库优化的物理基础。最后简单介绍了 DM642 的外设情况。

最后，在简单介绍了视频处理库的优化软件平台 CCS 之后，从 MPEG4 以及 H.264 编码器结构调整，存储器的优化，代码优化 3 个方面对编解码库优化方法进行了介绍。

由于时间和水平的限制，本文所进行的研究还不够全面。例如在帧层函数的优化还没有达到满意的效果，而且在函数手工汇编优化方面作进一步的研究。

致　谢

辍笔在即，思绪如潮。忆往昔求学生涯，得良师益友众人诸多资助，在此深表谢意。

首先，要真诚感谢刘云海导师，在我攻读硕士期间给予了热情的指导与无私关怀。刘老师治学严谨、学识渊博、思维敏锐，多年以来一直是作者学习的楷模。导师的悉心指导，严谨的治学作风和求是的科学态度使我受益匪浅，感受深刻。两年半的硕士学业和论文的圆满完成都离不开刘老师的指点和帮助。在此，向刘云海导师致以崇高的致意和由衷的感谢！

在硕士阶段的求学期间，得到了虞露老师、唐慧明老师、张朝阳老师、赵问道老师、陈慧芳老师、谢磊老师、史册老师等众多专家、学者的言传身教。感谢实验室的众位朝夕相处的伙伴，他们是胡琳、马海杰、刘斌兵、徐慧、颜凯、陈辉、汪燮彬、黄良飞、史昕亮、李蕾、楼佳佳、张芙蓉、刘先虎、蔡荣、林海涵、李殿福、黄飞等，这是一个充满团队协作精神、奋发向上的集体。感谢我在浙江大学认识的好朋友，感谢浙大两年半一起生活的室友，与他们相处给生活增添了无限温馨和快乐。对于他们的无私帮助和照顾表示深深的谢意。

最后特别感谢父母和亲人，他们的支持和关心是我不断进步的力量源泉，是我有能力、有信心面对困难，迎接挑战！

吕鸿波

浙江大学信息学院信电系

2006 年 02 月

参考文献

[1] Cheng Peng, Video Encoding Optimization on TMS320DM64x/C64x, http://www.ti.com 2004.8

[2] 钟玉琢 等 基于对象的数据压缩标准—MPEG4 及其校验模型。北京：电子工业出版社，2002.7

[3] TMS320DM642 Video/Imaging Fixed-Point Digital Signal Processor Data Manual，http://www.ti.com，2004.2

[4] Stewart Taylor Intel Integrated Performance Primitives ,How to Optimize SoftWare Applications Using Intel IPP ,Intel Press

[5] 徐华根 ，视频编码算法研究及其在 DM642 上的实现，2005

[6] Cache Usage in High-Performance DSP Applications With the TMS320C64x, http://www.ti.com，2001.12

[7] 雷国平，周琨，吉吟东 MPEG标准发展和研究综述，计算机工程，2003年第7期

[8] On the Implementation of MPEG-4 Motion Compensation Using the TMS320C62x，http://www.ti.com，1999.8

[9] Optimizing JPEG on the TMS320C6211 2-Level Cache DSP，http://www.ti.com，2000.12

[10] 黄铁军 数字音视频编解码技术标准AVS

[11] TMS320C6000 DSP Cache User's Guide，http://www.ti.com，2003.5

[12] TMS320C6000 EDMA IO Scheduling and Performance，http://www.ti.com，2003.3

[13] TMS320C6211 Cache Analysis，http://www.ti.com，1998.9

[14] TMS320C64x DSP Two-Level Internal Memory Reference Guide，http://www.ti.com，2004.8

[15] 赵德斌，陈耀强，高文 高质量图像压缩的自适应DPCM/DCT混合编码方法，计算机研究与发展 1998年第12期

[16] 余胜生，张剑，周敬利 基于H. 264标准的混合编码算法分析，计算机科学，2005第5期

[17] 刘殿文，李俭 图像编码技术研究动向 电讯技术 1995年第3期

[18] 王卫，蔡德钧，万发贯 一种多分辨率图像混合编码方案，通信学报，1995年第3期

[19] 付炜，林春雨，孟娟，景源 一种改善无损压缩性能的预处理及其理论分析，光电工程，

2004年第12期

[20] 杨胜天，仇佩亮 一组整数的几个实用编码方案，通信学报，2005年第7期

[21] 罗伟，张太镒，杨斌 AAC编码算法的快速实现，信号处理，2004年第6期

[22] 汪国有，张成兴，廖容，陈振学 MPEG一4 AAC实时音频编码器设计与实现研究，计算机与数字工程，2005年第8期

[23] TMS320C64x DSP Video Port/VCXO Interpolated Control (VIC) Port Reference Guide，http://www.ti.com，2004.8

[24] PAL/NTSC/SECAM video decoder with adaptive PAL/NTSC comb filter, VBI data slicer and high performance scaler Data Sheet，http://www.phlips.com，2004.3

[25] Intel LXT971A 3.3V Dual-Speed Fast Ethernet PHY Transceiver Data Sheet，http://www.intel.com，2002.8

[26] TMS320C6000 Chip Support Library API Reference Guide，http://www.ti.com，2004.8

[27] TMS320 DSP/BIOS User’s Guide，http://www.ti.com，2002.8

[28] Big-Endian Digital Signal Processing Library (DSPLIB) for TMS320C64x，http://www.ti.com，2003.9

[29] Video Encoding Optimization on TMS320DM64x/C64x，http://www.ti.com，2004.10

[30] TMS320C6000 Programmer’s Guide，http://www.ti.com，2002.8

[31] TMS320C6000 Assembly Language Tools User’s Guide，http://www.ti.com，2004.4

[32] TMS320C6000 CPU and Instruction Set Reference Guide，http://www.ti.com，2000.10

[33] 龚剑明 MPEG-4标准视频编码器在Blackfin 21535上的实现，电视技术，2003年第11期

[34] 王清亮，李中福，刘玉珊，郑黎 xvid视频编码技术，河南职业技术师范学院学报，2004年第3期

[35] 郭新军，王巧玲 基于MPEG4的视频采集与压缩，现代电子技术，2004年第1期

[36] 胡海涛，俞艳苹 基于TMS320CDSC21的MPEG4编码器二维DCT变换的实现，电子科技，2004年第2期

[37] Chun-Ling YANG，Lai-Man PO，Wing-Hong LAM A FAST H.264 INTRA PREDICTION ALGORITHM USING MACROBLOCK PROPERTIES，2004 International Conference on Image Processing

[38] Xuan Jing and Lap-Pui Chau An Efficient Inter Mode Decision Approach for H.264 Video Coding，2004 IEEE International Conference on Multimedia and Expo

[39] Jinghong Zheng and Lap-Pui Chau A Motion Vector Recovery Algorithm for Digital Video Using Lagrange Interpolation，IEEE TRANSACTIONS ON BROADCASTING, DECEMBER 2003

[40] Yong Ho Moon, Gyu Yeong Kim, and Jae Ho Kim An Improved Early Detection Algorithm for All-Zero Blocks in H.264 Video Encoding，IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 15, NO. 8, AUGUST 2005

[41] 王琪，贺玉文 基于对象的多媒体数据压缩编码国际标准——MPEG-4及其校验模型，科学出版社，2000

[42] 张云，林钟编著 “PC平台新技术MMX——应用编程实例”，东南大学出版社，1998

[43] 多媒体信息的传输与处理 北京，人民邮出版社，1999

[44] 丁贵广等 新一代静止图像编码JPEG2000概述，中国电子在线2004年8月5月

[45] 刘党辉，沈兰荪 DSP芯片及其在图像技术中的应用，测控技术，2001年第5期

[46] 王念旭 DSP基础与应用系统设计 北京航空航天大学出版社，2001年8月

[47] 张雄伟 DSP芯片的原理与开发应用，电子工业出版社，2003年2月

[48] 迅特公司 DM642硬件系统设计，2004年秋季TI&IMDSP技术研讨会

[49] 李波 基于通用DSP的多模式视频编码器，计算机学报，2004年3月

[50] Texas Instruments，Driver Example on the DM642 EVM，July 2004

[51] TMS320C6000 Optimizing C/C++ Compiler User's Guide，www.ti.com ，2001

[52] TMS320C6X C Source Debugger User Guide ，www.ti.com ，1999

[53] 张军 ASIC技术的特点与应用 信息技术，2001年第9期