

电子科技大学

硕士学位论文

基于JPEG标准的图像处理及其在MCF5329上的实现

姓名：周鹏斌

申请学位级别：硕士

专业：信息与通信工程

指导教师：林水生

20080601

## 摘 要

本文研究 JPEG 编解码算法在 DSP 上的实现技术。随着网络和嵌入式产品的飞速发展,数字图象在人们生活中的应用也越来越广泛,由于图像文件尺寸庞大,需要存储和传输的图像库数据也迅速增加。为解决日益增长的数据量与存储空间及网络传输带宽之间的矛盾,在图像数据存储和传输时必须对其进行压缩处理。在数字图像压缩领域,由于 JPEG 优良的品质,其应用是最为广泛的。

随着数字信号处理器(DSP)的功能的逐渐增强和价格的不断下降,DSP 已经被广泛应用到各种领域。DSP 比通用的 CPU 更适合做数字信号处理,在 DSP 上实现 JPEG 实时编解码具有内在的优势和一定的现实意义。

本文首先介绍了图像压缩技术的原理、JPEG 标准和基于 DSP 的实时信号处理系统,然后在 Freescale ColdFire 32 位微处理器 MCF5329 上设计并实现了 JPEG 编解码算法。JPEG 编码算法主要由色彩空间转换、亚抽样、DCT 变换、DCT 系数量化和哈夫曼编码等几个部分组成。重点结合 MCF5329 的流水线操作和并行操作特征及离散余弦变换算法原理,利用乘累加器合并加法运算和乘法运算高效快速地实现了一种基于 DSP 乘法累加单元的 DCT 快速算法,并对几个关键模块进行了优化。整个编码算法能够达到实时压缩的目标。

相应地, JPEG 解码算法主要由哈夫曼解码、DCT 系数反量化、IDCT 变换、内插和色彩空间转换等几个部分组成。利用 EMAC 单元实高效地实现了反离散余弦变换快速算法,并对几个关键模块进行了优化。整个解码算法能够达到实时解压显示的目标。

编解码完成后,对编解码器的功能和性能作了测试和对比分析。对一幅大小为  $240 \times 320$  的 24 位图片进行编解码,所需时间分别为 0.23s 和 0.12s,达到了实时编解码的目的。分析结果表明对 DCT 算法和 IDCT 算法部分实现和优化较为成功,对哈夫曼编解码部分的优化则相对欠缺,这也是 JPEG 编解码软件需要加以改进的地方。

关键词: JPEG, MCF5329, 编码器, 解码器, 优化

## **Abstract**

The direction of the paper is the realization of JPEG coder and decoder technology under DSP. With the rapid development of network and embedded system, digital images have been used widely in modern society. With the huge size of the image, the image database increases rapidly accordingly. In order to resolve the problem between increasingly data and memory storage, the image data must be compressed in storage and transmission. The JPEG compression standard is most popular in the field of digital image processing because of the good quality of JPEG. Digital signal processor (DSP) is more suitable for digital signal processing than general CPU. Thus there will be of great benefit to employ DSP to real-time image coder and decoder.

In the dissertation, the principle of image coding, the JPEG system and the real-time signal processing system based on DSP are introduced first. Then the design of a JPEG real-time image coder and decoder system based on Freescale ColdFire MCF5329 is discussed. The coder system is mainly consist of color space transformation, down-sampling, DCT transform, quantization and huffman coder. Combining the principles of pipelining and parallelism of DSP with DCT theory, we concentrate on the use of Enhanced Multiply Accumulate Unit of MCF5329 by merging the operations of addition and multiplication, and realize 8×8 two dimensions of DCT with one dimension of DCT efficiently. Tests show that the software meets the requirement of real-time coder.

Accordingly, the JPEG decoder is mainly consist of huffman decoder, dequantization, IDCT transform, up-sampling and color space transformation. We realize the fast IDCT algorithm efficiently and optimize some key modules. Tests show that the software meets the requirement of real-time decoder.

Finally, the dissertation summarizes the shortage of the system, and take the method to modify and introduce the prospect of the system in the future.

**Keywords:** JPEG, MCF 5329, coder, decoder, optimization

## 图目录

图 2-1	基本系统编解码结构图 .....	12
图 2-2	DCT 与 KLT 系数的方差分布 .....	14
图 2-3	8×8 像素值矩阵和 8×8 系数矩阵 .....	15
图 2-4	DCT 系数的“Z”型排列和 DCT 系数序号 .....	22
图 2-5	行程编码 .....	23
图 3-1	MCF 5329 开发板实物图 .....	29
图 3-2	MCF5329 结构图 .....	30
图 3-3	ColdFire 处理器核的流水线结构 .....	31
图 3-4	ColdFire EMAC 功能框图 .....	32
图 3-5	ColdFire MAC 寄存器 .....	32
图 3-6	ColdFire V3 用户编程模式寄存器 .....	33
图 4-1	编码流程图 .....	40
图 4-2	优化前后流程图对比 .....	42
图 4-3	h2v2 亚抽样 .....	44
图 4-4	交流系数的编码 .....	50
图 4-5	ENCODE_R,ZZ(K) .....	51
图 4-6	Lena.bmp (512×512, 24 位色, 大小为 768KB) .....	53
图 4-7	Q=100, 文件大小为 179KB, 压缩比为 4.3 .....	53
图 4-8	Q=50, 文件大小为 24.3KB, 压缩比为 31.6 .....	54
图 4-9	Q=20, 文件大小 13.7KB, 压缩比为 56.1 .....	54
图 5-1	JPEG 解码器基本流程图 .....	58
图 5-2	基于 MCF5329 核微处理器解码流程图 .....	59
图 5-3	MINCODE()、MAXCODE()、VALPTR()的生成 .....	60
图 5-4	DECODE 过程 .....	61
图 5-5	NEXTBIT 过程 .....	62
图 5-6	RECEIVE(S)和 Huff_EXTEND(C,S) .....	62
图 5-7	交流系数的哈夫曼解码 .....	63
图 5-8	Decode_ZZ(K) .....	64
图 5-9	h2v2 内插 .....	69
图 5-10	解码后的 JPEG 图像在 LCD 上显示 .....	70

# 表目录

表 1-1	图像编码方法分类 1 .....	2
表 1-2	主观测试分级标准 .....	4
表 2-1	各种快速 DCT 算法 .....	17
表 2-2	亮度和色度量化矩阵 .....	22
表 4-1	乘累加指令 .....	45
表 4-2	DSP 计算 8 点 DCT 时间 .....	47
表 4-3	定点亮度量化表 .....	48
表 4-4	定点色度量化表 .....	48
表 4-5	优化前后编码时间比较 .....	55
表 4-6	各模块优化后效果 .....	55
表 4-7	优化前后编码器所需存储空间 .....	56
表 4-8	不同压缩比率下图像的 PSNR 值 .....	57
表 5-1	DSP 计算 8 点 IDCT 时间 .....	68

## 缩略词表

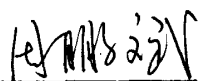
英文缩写	英文全称	中文释义
ABAC	Adaptive Binary Arithmetic Coding	自适应二进制算术编码
ADCT	Adaptive Discrete Cosine Transformation	自适应离散余弦变换
AGEX	Address Generation, Execute	地址生成和指令执行
BSPC	Block Separated Progressive Coding	块分离的累进编码
DCT	Discrete Cosine Transformation	离散余弦变换
DFT	Discrete Fourier Transformation	离散傅立叶变换
DHT	Define Huffman Table	定义哈夫曼表
eDMA	Enhanced Direct Memory Access	增强直接存储器存取
DNL	Define Number of Lines	定义行数
DPCM	Differential Pulse Code Modulation	差分脉冲编码调制
DQT	Define Quantization Table	定义量化表
DRI	Define Restart Interval	定义重启间隔
DSOC	Decode&Select, Operand Fetch	指令译码与操作数选择
DSP	Digital Signal Processor	数字信号微处理器
EMAC	Enhanced Multiply-Accumulate Unit	增强型乘累加单元
EOB	End of Block	块结束
EOI	End of Image	图像结束
FEC	Fast Ethernet Controller	快速以太网控制器
FIFO	First Input First Output	先入先出队列
GDB	Background Debug Mode	背景调试模式
IDFT	Inverse Discrete Fourier Transformation	反离散傅立叶变换
IFP	Instruction Fetch Pipeline	取指令流水
ISO	International Organization for Standardization	国际标准化组织
ITU	International Telecommunication Union	国际电信联盟
I <sup>2</sup> C	Inter Integrated Circuit	集成电路内部总线
JBIG	Joint Bilevel Image Group	二值图像编码标准
JFIF	JPEG File Interchange Format	JPEG 交换格式

# 缩略词表

JPEG	Joint Photographic Experts Group	联合图像专家组
KLT	Karhunen-Loeve Transformation	卡胡南-列夫变换
LTIB	Linux Target Image Builder	Linux 目标镜像生成
MACSR	Multiply-Accumulate Unit Status Register	乘累加单元状态寄存器
MASK	Multiply-Accumulate Unit Mask Register	乘累加单元屏蔽寄存器
MCU	Minimum Code Unit	最小编码单元
MPEG	Moving Picture Expert Group	运动图像专家组
MIPS	Millions of Instruction Per Second	每秒执行百万条指令
MSE	Mean Square Error	均方误差
OEP	Operand Execution Pipeline	操作数执行流水线
PLL	Phase-Locked Loops	锁相环
PSNR	Power Signal-to-Noise Ratio	信号噪声功率比
PWM	Pulse Width Modulation	脉宽调制
QSPI	Queued Serial Peripheral Interface	队列串行外围接口
RLE	Run Length Encoding	行程编码
SRAM	Static Random Access Memory	静态随机存储器
SOI	Start of Image	图像开始
SOS	Start of Scan	扫描开始
SSI	Synchronous Serial Interface	同步串行接口
UART	Universal Asynchronous Receiver/transmitter	通用异步接收器/发送器
USB	Universal Serial Bus	通用串行总线
VLC	Variable Length Code	变长码
VLI	Variable Length Integer	变长整数
ZRL	Zero Run Length	零行程

## 独创性声明

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

签名: 

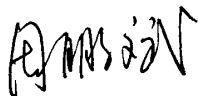
日期: 2008年5月22日

## 关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

(保密的学位论文在解密后应遵守此规定)

签名:



导师签名:



日期: 2008年5月22日



## 第一章 绪论

### 1.1 论文背景

二十一世纪是一个信息化的时代，信息技术飞速发展，信息的内容也从简单的文字、语音发展到高清晰的图像、动画和实时的音视频。多媒体技术的发展大大提高了人们的生活质量，网上购物、远程医疗、在线电影和 3D 游戏已经进入千家万户，数码相机、可视手机、MP4 播放机等多媒体设备也越来越普及。然而，大量的多媒体信息带来了“信息爆炸”，大量数据的存储和传输成为多媒体技术发展中的一个棘手的瓶颈问题。单纯用扩大存储器容量、增加通信干线的传输率的方法是不现实的，而多媒体信源压缩编码是个行之有效的方法。在一定的图像保证度的前提下，对图像数据进行压缩后再存储和传输，既节约了存储空间，又提高了通信干线的传输效率。

本论文主要对某半著名导体公司研究基金项目的实时图像处理部分作阐述，在 MCF5329 嵌入式平台上实现  $240 \times 320$  大小的图片的实时压缩和解压缩。虽然 JPEG2000 压缩新标准已经公布，但由于该项目对压缩比要求不高，而在低压缩比的情况下 JPEG 和 JPEG2000 的性能相差不大，但 JPEG 的算法复杂度远远低于 JPEG2000，故选取 JPEG 压缩标准。在实现 JPEG 正确编解码功能的基础上，对 JPEG 编解码程序进行了指令级和算法级的优化，使其性能有了较大的改善和提高，以满足系统的应用需求。

### 1.2 图像压缩编码技术

#### 1.2.1 图像数据压缩的原理

信息论的观点认为图像中总是或多或少地含有自然冗余度，这些冗余度既来自信源本身的相关性，又来自信源概率分布的不均匀性。图像压缩编码是在对数字图像进行大量统计分析，充分了解和掌握图像信息统计特性的基础上，充分利用人眼的视觉效应和图像本身相关性强的特点，寻求消除和减少相关性，或改变

图像信源概率分布不均匀性的方法和手段，实现图像数据的压缩。消除或减少图像的冗余度是实现图像压缩的基本依据。图像数据的冗余有以下类型<sup>[1]</sup>：

- (1) 空间冗余
- (2) 时间冗余
- (3) 信息熵冗余
- (4) 结构冗余
- (5) 知识冗余
- (6) 视觉冗余
- (7) 局部相似性冗余
- (8) 图像区域的相同性冗余
- (9) 纹理统计冗余

1.2.2 图像数据压缩编码技术的分类

根据压缩原理进行划分，编码方法可以分为四大类，如表 1-1<sup>[2]</sup>所示。这些方法既适用于静止图像编码，也适用于视频图像信号编码。

表1-1 图像编码方法分类 1

算法分类	编码方法举例	主要特点
熵编码	哈夫曼编码	熵编码又称为统计编码，是建立在图像统计特性基础之上的
	算术编码	
	RLE 编码	
预测编码	DPCM 编码	依据某一模型，根据以往的样本值对于新样本值进行预测编码
	运动补偿法	
变换编码	DCT 变换编码	将空域中描述图像的数据经过变换，转换到变换域中进行描述，达到改变能量分布的目的，实现对信源图像数据的有效压缩
	DFT 变换编码	
	小波变换编码	
混合编码	JPEG 编码	主要包括静止图像和运动图像的压缩标准系列
	MPEG 编码	

根据解码后的图像数据与原始图像数据是否完全一致，可分为有损压缩编码和无损编码。有损压缩编码是指对图像进行解压后重新构造的图像与原始图像存在着一定的误差。无损编码是指对图像进行解压后重新构造出来的图像与原始图

像之间完全相同。

有损压缩利用了图像信息本身所包含的许多冗余信息，例如视觉冗余和空间冗余。由于有损压缩一般情况下可获得比较好的压缩比，因此在对图像的质量要求不苛刻的情况下一般应选择有损编码。

为了实现更高的编码效率，具体编码时并不一定是孤立地、单一地使用一种方法，往往需要各种方法的交叉使用。

### 1.2.3 图像压缩系统的性能评价和度量

评价图像压缩技术和系统的性能，主要考虑三个关键参数<sup>[3]</sup>：

- (1) 压缩比——在达到所要求的恢复图像质量的条件下，压缩比越高越好。
- (2) 图像质量——图像质量的评估包括两个方面：一个是图像保真度，即被压缩处理的待评价图像与原始图像之间的偏离程度；另一个是易懂度，它是指图像向人或机器提供信息的能力。
- (3) 压缩/解压的简易性——包括压缩/解压的速度，软件和硬件的开销或占用的资源。

#### 1.2.3.1 图像压缩效率的评定标准

图像压缩系统是可以由冗余度、编码效率、和压缩比等指标来衡量信源特性和编解码设备性能的。设原始图像的平均码长为 $\bar{L}$ ，熵为 $H(S)$ ，压缩后图像的平均码长为 $\bar{L}_c$ ，则定义：

$$\text{冗余度: } R = \frac{\bar{L}}{H(S)} - 1$$

$$\text{编码效率: } \eta = \frac{H(S)}{\bar{L}} = \frac{1}{1+R}$$

$$\text{压缩比: } C_r = \frac{\bar{L}}{\bar{L}_c}$$

其中 $H(S) = -\sum_{k=1}^M P_k \log_2 P_k$ ， $P_k$ 为像素点灰度概率。

从上面公式可以知道，图像压缩效率主要由压缩比 $C_r$ 来衡量，压缩比越高，图像压缩率越大，反之亦然。

### 1.2.3.2 图像质量的评定标准

在对一幅图像进行压缩处理过程中，由于包括的环节很多，最终恢复出来的图像的性能的优劣到底如何，这就需要有一个对图像质量进行评判的标准。图像质量的含义最主要是包含了两层意义。第一就是是恢复出来的图像的逼真度，即被压缩处理的待评价图像与原始图像之间的偏离程度；第二就是图像的可懂度，它是指图像向人或机器提供信息的能力。一般来说，图像质量评价方法分为主观评价方法和客观评价方法。

表1-2 主观测试分级标准

级别	质量	损伤	比较
5	优	不能觉察	好得多
4	良	刚觉察，不讨厌	相同
3	中	有点讨厌	略坏
2	次	很讨厌	坏
1	劣	不能用	坏得多

图像的主观评价是由评价者直接对一幅图像进行观察，按照一定规则并根据自己的经验对图像的优劣作质量判断，从感觉上去度量其失真程度，给出质量评价级别，对所有评价者给出的分数进行加权平均，所得结果即为主观评价结果，如表 1-2 所示。这种评价结果必然符合人的视觉感受。但这种主观感受一方面不能用数学模型对其进行描述，不能直接用于视频图像压缩编码过程中的质量评价与控制；另一方面，主观评价容易受到评价者的主观因素的影响，如年龄、性格、教育程度、背景以及评价时的心情等。因而这种评价方法在实际应用中受到了很大的限制，甚至根本不适合某些应用场合。

人的视觉能力有限，在一定的条件下，一定的视频失真人眼根本无法觉察出来，所以对视频质量的主观评价和客观度量不同点在于人眼的视觉特性，其中之一是视觉掩蔽效应。由于视觉掩蔽效应，一定的视频失真被掩蔽掉了，从主观评价的角度看，这部分损伤并没有计入视频失真之内。而从客观度量评价的角度看，任何偏离原始视频的误差都被认为是失真，这是主客观评价不一致的主要原因。

所谓客观方法，就是定义一个数学公式，然后对待评价的图像进行运算，得

到一个唯一的数字量作为测度结果，这种方法最常用于对图像的逼真度评价。对于静止图像的客观评价是用重构图像偏离原始图像的误差来衡量的，常用的有均方误差（MSE）和峰值信噪比（PSNR）。

(1) 均方误差的定义为：

$$MSE = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [f(i, j) - f'(i, j)]^2 \quad (1-1)$$

其中，M,N 是图像宽和高的像素点数；

$f(i, j)$  是原始图像的灰度值；

$f'(i, j)$  是重建图像的灰度值。

(2) 峰值信噪比的定义为：

$$PSNR = 10 \log_{10} \frac{f_{\max}^2}{MSE} \quad (1-2)$$

其中  $f_{\max}$  代表像素点最大取值。

有时候评价的结果往往和人的视觉感受不一致。这主要是由于误差的均匀程度造成的，因为均方误差（MSE）和峰值信噪比（PSNR）是从总体上反映原始图像和重构图像的差别，并不能反映局部差别。一般来讲，误差均匀时视觉效果较好，反之视觉效果相对较差。一般的情况下，对图像的质量测度都可以用峰值信噪比来评判，但有时在特殊的情况下，峰值信噪比对图像质量的评判的结果可能和主观上评判的结果不相符合。

### 1.3 图像压缩技术的国际标准及发展

近年来，国际标准化组织（ISO）和国际电信联盟（ITU）先后制定了多个图像/视频编码的标准。如二值图像编码标准（JBIG）、静止图像编码标准（JPEG 和 JPEG2000），以及各种面向视频序列的压缩标准。下面简单介绍静止压缩编码的标准。

### 1.3.1 静止图像压缩标准

#### 1.3.1.1 JBIG 标准

JBIG(Joint Bilevel Image Group)<sup>[4]</sup>是联合二值图像组的缩写,它是适用于二值图像压缩的标准。JBIG 可以支持很高的图像分辨率。JBIG 采用累进操作方式,可以使具有不同分辨率的图像设备使用同一个压缩图像,可以方便地在一组图像中浏览,非常适于在分组网中传输。JBIG 采用无损压缩技术,但它的压缩率比目前的传真标准(CCITT G-3, G-4 标准)高得多。

JBIG 的编码器可以分解为 D 个相同的差分层编码器串联,最后一个为底层编码器,其中 D 是累进参数,可任意选择,一般为 4 至 6。当 D=0 时,JBIG 进行非累进图像压缩。差分层编码器和底层编码器的核心是二个自适应算术编码器。在差分层编码器中还含有把分辨率降低一半的功能。JBIG 的解码过程与编码过程正好相反。

JBIG 的压缩率可达到 10:1。虽然 JBIG 是二值图像的编码标准,但也可以对含灰度值的图像或彩色图像进行无失真压缩,这时 JBIG 对图像的每个比特面作压缩变换。

#### 1.3.1.2 JPEG 标准

JPEG(Joint Photographic Expert Group)是 CCITT SGVIII(国际电信电话咨询研究委员会研究组 VIII)的 CCIC(Common Component for Image Communication)和 ISO JTC1/SC29/WG10 的联合成立的一个组织,设立于 1986 年 3 月,该组织进行了正式的图像编码标准化活动,并于 1988 年 1 月选定 ADCT(Adaptive Discrete Cosine Transformation)作为标准草案的基础,在 1988 年 9 月,这种静态图像的国际标准编码方式被定为 JPEG 算法<sup>[5]</sup>。

图像的编码压缩主要用于图像的存储和图像通讯。数字图像的信息量大,而且采用数字是通讯制式时占用的频带宽,所以编码压缩图像可以节省存储空间,通讯时减少占用的频带宽度。JPEG 算法是多种编码方法的综合,有这样一个标准算法的好处在于:

- (1) 通用性高,能在各种设备中广泛的使用;
- (2) 压缩效率高,充分利用了人眼视觉的低通滤波特性;

- (3) 恢复的图像质量好, 可视性和可懂性都较好;
- (4) 是多种编码方法的综合, 考虑了可逆编码和非可逆编码, 串行编码和并行编码;
- (5) 易于实现, 执行速度较快。

### 1.3.1.3 JPEG2000 标准

为了弥补目前标准的不足, 适应 21 世纪图像压缩的需要, 早在 1997 年, ISO/ITU-T 组织下的 IEC JTC1/SC29/WG1 小组便开始着手制定新的静止图像压缩标准——JPEG2000。并于 2000 年 12 月推出标准的第一部分, 即 JPEG2000 Part I (ISO 15444-1)<sup>[6]</sup>。与 JPEG 不同, JPEG2000 基于小波变换, 采用当前最新的嵌入式编码技术, 在获得优于目前 JPEG 标准压缩效果的同时, 生成的码流有较强的功能, 可应用于多个领域。

JPEG 2000 的主要目标如下:

JPEG2000 标准致力于建立一个能够适用于具有各种不同特性(自然图像、科学图、医学图、遥感图像、文本等)不同类型(单色图、灰度图、真彩色)的图像, 允许在不同工作模式(客户/服务器、实时传播、电子图书馆等)下使用统一的标准。

- (1) 优异的低比特率性能;
- (2) 连续色调和二值图像的压缩;
- (3) 压缩大幅图像;
- (4) 无损压缩和有损压缩;
- (5) 累进传输;
- (6) 随机码流访问和处理;
- (7) 输出比特率的可控性;
- (8) 对比特错误具有鲁棒性。

### 1.3.2 图像压缩系统的应用和发展

数字图像压缩与信息的传输(如各种媒体的局域或广域通信)、传播(如数字电视)和存储息息相关。换言之, 数字图像压缩技术是信息产业中三大支柱产业一

通信、计算机和消费类电子产品(主要以图像、视频产品为主要代表)的共性核心技术之一,也是三大网络—电信网、计算机网和电视网逐步融合,逐步走向三网合一的技术基础之一。

自上个世纪 80 年代以来,图像/视频压缩编码技术及其标准化工作取得了一系列重大成就。如 ITU 的 H.261、H.263、H.264 和 ISO/IEC 的 JPEG、JPEG2000、MPEG-1、MPEG-2、MPEG-4、MPEG-7 以及 MPEG-21 等,成为信息产业中不可缺少的关键技术。

目前,图像编码的最新发展有以下几个方面<sup>[7]</sup>:

多分辨率处理(Multiresolution Processing),在这方面值得一提的是 P.J.Burt 和 E.H.Adelson 提出的拉普拉斯金字塔编码法(The Laplacian Pyramid Code)及 T.R.Fischer 提出的塔形矢量量化(Pyramid Vector Quantization)等方法;

时频域分析(Time-Frequency Analysis),如小波变换分析(Wavelet Transform Analysis),多维子带编码(Multidimensional Subband Coding),基于视觉模型的编码(Coding Based on the Signal Processing)。小波变换比传统的傅立叶变换有良好的局部时频域特性,有“数学显微镜”之称,基于小波变换的图像编码很有发展前途。

## 1.4 主要工作及论文结构

本论文以国际电信联盟(ITU)和国际标准化组织(ISO)联合组成的联合图像专家组(JPEG)公布的 ISO10918-1,即 JPEG 图像压缩编码标准为基础,在 MCF5329+ $\mu$ Clinux 软硬件环境下,实现了 JPEG 编解码器功能,并结合 MCF5329 指令特点对关键模块进行了优化,达到了图像实时编解码的目的。

主要工作包括以下几方面:

- (1) 设计并实现了 JPEG 编码器。改进了 JPEG 编码流程,实现了一种基于 DSP 乘法累加单元的 DCT 快速算法,对色系变换模块、亚抽样模块、DCT 系数的量化模块分别进行了优化。
- (2) 设计并实现了 JPEG 解码器。改进了 JPEG 解码流程,实现了基于 DSP 的快速 IDCT 算法——AA&N 算法,对哈夫曼解码模块、量化模块、内插模块和色彩空间转换模块分别进行了优化。



(3) 对编解码器的性能作了测试与分析。

本论文共分为六章，各章的内容安排如下：

绪论部分介绍了本文的研究背景，图像压缩编码的原理、评价图像压缩质量的性能指标及相关静止图像压缩编码的国际标准，论文的主要内容以及结构安排。

在第二章中，首先对 JPEG 压缩系统的研究背景和标准化制订历史进行了简要的介绍，然后详细介绍了 JPEG 压缩算法的原理。对 DCT 变换、DCT 系数量化、熵编码、亚抽样与内插进行了深入地分析和阐述。

第三章主要介绍了 JPEG 编解码算法实现的具体软硬件平台。对硬件开发平台中的处理器 MCF5329 的特点、结构及 ColdFire V3 内核进行了详细的介绍和分析。最后对软件开发平台 LTIB 作了简要介绍。

第四章研究设计了基于 MCF5329 的 JPEG 编码软件，对编码软件中的关键部分进行了优化。在改进后的系统总体框架基础上，按照 JPEG 编码流程逐步实现了编码软件的各个模块。在 DCT 变换部分应用了一种基于 DSP 乘累加单元的 DCT 快速算法，对 DCT 系数量化、色系变换和亚抽样算法进行了 DSP 优化。最后对编解码器进行了测试与结果分析。

第五章研究设计了基于 MCF5329 的 JPEG 解码软件，对解码软件中的关键部分进行了优化。在改进后的系统总体框架的基础上，按照 JPEG 解码流程逐步实现了解码器的各个模块。在 IDCT 部分实现了快速算法 AA&N 算法的 DSP 代码，对内插函数代码和色彩空间转换代码进行了优化。最后对解码器进行了测试与结果分析。

最后一章对全文进行了总结，指出了目前存在的问题，并讨论进一步的研究的方向和关键技术难点。

## 第二章 JPEG 标准的算法分析

### 2.1 JPEG 算法与 JPEG 小组简介

1986 年, CCITT 和 ISO 两个国际组织建立联合图片专家组(Joint Photographic Experts Group, 简称 JPEG), 其任务是建立第一个适用于连续色调图像压缩的国际标准。JPEG 的目标是开发一种用于连续色调图像压缩的方法, 这种方法必须满足以下 4 点要求<sup>[8]</sup>:

- (1) 应用当前的先进图像压缩技术。在保证压缩率的同时, 图像质量要好, 即失真程度要在一定的范围之内。编码器的参数中应该包括控制压缩比和图像质量的成分。
- (2) 适用于所有的连续色调图像, 不对图像的尺寸、彩色空间和像素纵横比等特性进行限制, 也不应对图像的场景(如复杂度、彩色范围或统计特性等)有任何要求。
- (3) 具有适中的计算复杂度, 从而使得压缩算法既可以用软件实现, 也可以用硬件实现, 并具有较好的性能。
- (4) 具有下述四种操作模式: 顺序编码、累进编码、无失真编码和层次编码。

方程部分(下一个) 根据 JPEG 的要求, 研究人员提出了 12 种建议方案。1986 年 6 月在哥本哈根电话公司研究实验室对最初的 12 个方案进行评估, 评估的根据主要是图像质量, 最后选出如下 3 种方案:

- (1) 自适应离散余弦变换(Adaptive Discrete Cosine Transform, 简称 ADCT)
- (2) 自适应二进制算术编码器(Adaptive Binary Arithmetic Coder, 简称 ABAC)
- (3) 块分离的累进编码(Block Separated Progressive Coding, 简称 BSPC)

在这次测试中, 对 10 种有效的建议算法进行了测试。测试时使用 4 个标准测试图像, 要求建议的算法把图像压缩率设置为 0.25, 0.75 和 4 位/像素。

JPEG 专家组对所产生的 120 幅图像的质量进行了评价, 结果是:ADCT 建议在 0.75 位率时图像质量最好, 在这种情况下, 对多种应用都能产生较好的图像质量。

ABAC 建议在 0.25 位率时效果最好, 它更适用于诸如图像数据库之类需要快速查找的应用。

继 1986 年的第 1 轮测试后, 于 1988 年 1 月又对剩下的 3 个建议 (ADCT 建议、ABAC 建议和 BSPC 建议) 进行了第 2 轮测试。这次 JPEG 专家组对参加测试的 3 个算法提出了更高的要求。要压缩比率为 0.08, 0.25, 0.75 和 2.25 位/像素, 选择了 5 幅新的测试图像。根据这个规定, 共产生 60 幅压缩图像( $3 \times 4 \times 5$ )。JPEG 专家组的 31 位专家对这 60 幅图像分别进行打分, 并且为了保证公正性, 专家们在打分的时候并不知道特定的图像所用的算法种类。为了排除随机性, 打分过程共进行了 4 轮。结果是 ADCT 为最佳方案。

根据 1988 年 1 月的测试结果, JPEG 专家组一致同意以 ADCT 为基础提出一个 ISO 标准草案, 该标准草案于 1990 年 3 月得到通过, 并把这个标准草案命名为 JPEG, 1992 年 JPEG 正式成为国际标准, 编号为 ISO/IEC 10918。

JPEG 标准中定义了两种不同性能的系统: 基本系统 (Baseline System) 和扩展系统 (Extended System)。基本系统采用顺序工作方法, 在熵编码阶段使用 Huffman 编码方法来降低冗余度, 解码器只存储两个 Huffman 表。扩展系统提供增强功能, 它是基本系统的扩展, 使用累进方法工作, 编码过程采用自适应的算术编码。之所以在 JPEG 中定义两种性能不同的系统, 主要是考虑到 JPEG 设备的兼容性和实现的方便性。每个标准解码器都应该能解释用基本系统编码方法编码的数据。在扩展中, 仅当编码器和解码器都配置相应的选项时才具有增强功能。

### 2.1.1 基于 DCT 的基本系统

基本系统是基于 DCT 的顺序编解码算法, 它要求像素的各个分量都具有 8 位 (bit) 精度。以下分别讲述它的三个特点。

利用 DCT 进行变换编码, 不必假设图像模型, 所以此方法使用于各种图像的压缩。对 DCT 系数的线性化可以考虑人的视觉对各频率分量的心理阈值, 从而保证图像质量。可以通过线性地改变 DCT 系数的量化阶距, 来控制图像的质量, 或控制压缩比。DCT 的系数量化后适合于熵编码。基本系统只支持顺序编解码, 不支持渐进编解码。顺序编码只需要对图像从顶到底处理一次, 每次一行。这种操作可以使编解码设备的缓冲降到最小限度, 从而降低了设备的价格。

基本系统要求像素的各个色彩分量都是 8bit; 对于那些精度低于 8bit/像素分量

的图像，可以通过字节（byte）内的左移来满足这个要求。

基本系统编解码结构框图如图 2-1 所示，左半部分为编码，右半部分为解码。

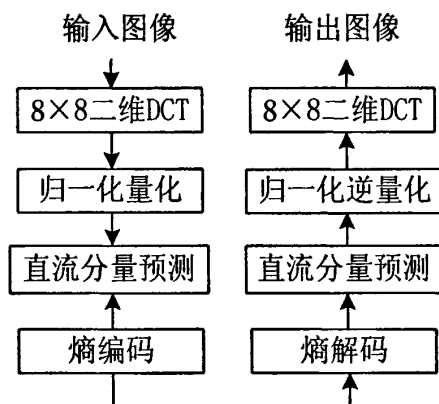


图2-1 基本系统编解码结构图

编码器首先将图像分割成  $8 \times 8$  的数据块，然后作  $8 \times 8$  的正向 DCT 变换，DCT 系数经线性量化后，预测出直流分量，最后利用变长码表（VLC）进行熵编码。解码器的操作顺序与编码相反，并在每一步操作中都做与编码器相反的操作。

## 2.1.2 基于 DCT 的扩展系统

扩展系统与基本系统一样，都是基于 DCT 变换域编码，但是基本系统只是扩展系统的一个子集。扩展系统提供三种额外的功能：算术编码，高精度输入，以及渐进编码操作。以下从这三方面分别讲述。

### (1) 算术编码（Arithmetic Coding）

基本系统中采用利用变长码的熵编码技术，而在 Extended System 中可用算术编码完成同样的功能。算术编码自适应于图像的统计特性，它的压缩比比采用固定变长码表的霍夫曼编码高出 5~10%。但是，更多的人认为算术编码太难实现。

### (2) 输入图像的高精度

扩展系统支持 12bit/像素分量的输入精度。在此精度下，DCT 变换必须有更高的精度，DCT 系数必须用 15bit 来表示；此外，量化表、变长码表也要相应地增高精度或加以扩展，以满足高精度的要求。

### (3) 渐进编解码（Progressive Operation）

渐进编解码是指将图像的质量分成若干等级，每一级对应一次编解码。首先进行质量最低一级编解码，接着依次进行质量略高一级的编解码，直到所要求的最高质量为止。

渐进编码用于传输速率较低的信道上是非常有用的。例如在比较忙碌的信道上，可以利用渐进编码先发送较低质量的图像，使接收端解码后可以看到图像的大致轮廓；而后依次发送质量较高一点的图像，使接收端的解码输出不断丰富原有图像的细节，直到预先定义的质量达到为止。这样可以弥补顺序编解码操作中解码输出前单纯地等待的缺陷。

实现渐进编解码有两种方法：频谱选择和逐渐逼近的方法。而渐进编解码中的 DCT 变换及其反变换则与顺序编解码中的一样。而且当渐进编解码的最后一级完成后，图像质量与顺序编解码的输出图像质量是相同的。以下分别讲述频谱选择和逐渐逼近的渐进编解码原理，请注意：这两种方法可以单独使用，也可以同时使用。

频谱选择是指把 DCT 系数分成若干个频带，每一级编解码都针对整个图像所有  $8 \times 8$  数据块中的一个频带。习惯上是先编码较低频带，后编码较高频带。DCT 系数中的直流分量往往作为一个单独的频带发送，而不是把它和交流分量合并在一个频带内。

逐渐逼近的方法是指首先发送所有  $8 \times 8$  数据块中的 DCT 系数的高位，然后依次发送其高位及低位。这样接收端首先收到 DCT 系数的高位，相当于 DCT 系数的一个近似值，随着次高位以及低位数据的接收，这种近似引起的误差不断减少，故称这种方法为逐渐逼近。

频谱选择和逐渐逼近两种方法可以嵌套使用。例如可在逐渐逼近的一级中，引入频谱选择，先发送低频 DCT 系数的某位，再发送高频 DCT 系数的某位；也可以在频谱选择的某一个频带内，引入逐渐逼近，先发送这一频带内 DCT 系数的高位，再发送这一频带内的 DCT 系数的低位。

### 2.1.3 JPEG 算法中的空间域预测编码

JPEG 算法中，除了基于 DCT 的一类编码方法外，主要是基于 DPCM 的空间域预测编码。DPCM 编码后发送的值是像素值与其预测值的残差。预测值是由像素的相邻像素值加权相加得到的，一般取正上方、左方和左上方这三个相邻像素

的加权平均值。

由于编解码采用同样的预测器，所以 DPCM 是一种无损压缩方法。而且，精度在 16bit/像素分量以下的输入图像都可以采用 DPCM 无损压缩。

## 2.2 JPEG 算法分析

考虑到实际课题原因，我们这里只介绍基本系统的原理。

### 2.2.1 离散余弦变换

#### 2.2.1.1 离散余弦变换简介

实际应用中的变换编码常采用离散余弦变换 DCT，DCT 相对于 KLT 可以看作一种次佳变换，它的结果很接近于 KLT。图 2-2 给出了 DCT 与 KLT 的变换域系数方差分布曲线，可见 KLT 的变换域系数方差比 DCT 更集中，而 DCT 是 KLT 的很好近似。

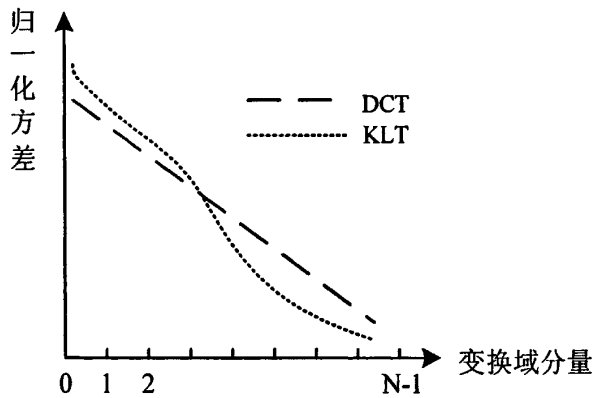


图2-2 DCT 与 KLT 系数的方差分布

$N$  个点的一维序列 ( $0 \leq i \leq N-1$ ) 的 DCT 定义如下：

$$y_0 = \sum_{i=0}^{N-1} x_i \quad (2-1)$$

$$y_k = 2 \sum_{i=0}^{N-1} x_i \cos[\pi(2i+1)k / 2N], 1 \leq k \leq N-1 \quad (2-2)$$

其逆变换 IDCT 定义为:

$$x_i = (1/N) \sum_{k=0}^{N-1} y_k \cos[\pi(2i+1)k/2N], 0 \leq i \leq N-1 \quad (2-3)$$

在图像编码中, 用 DCT 做变换而不用 KLT 或 FFT 主要有以下几个原因<sup>[9]</sup>:

- (1) DCT 比 KLT 的频域概念更直觉, 容易根据人类视觉的频率分析机理, 即视觉心理阈值来控制变换域系数的量化噪声。
- (2) DCT 实际上是源于 DFT, 但 DCT 省去了复数运算, 故运算量和数据量都较小, 而且不用传送基矢量。
- (3) DCT 受加窗的影响比 DFT 小, 故频谱畸变也小。

从统计特性来看, DCT 接近 KLT, 而其变换效率则高于 DFT。

### 2.2.1.2 二维离散余弦变换

基本系统中的 DCT 要求输入数据是一个  $8 \times 8$  的矩阵, 且每个矩阵元素具有 8bit 的精度, 范围从 -128 到 127, 故 DCT 变换前, 像素值先要减去 128。所谓  $8 \times 8$  的二维 DCT 是指将  $8 \times 8$  的像素值矩阵变换成  $8 \times 8$  系数矩阵。像素值矩阵和 DCT 系数矩阵见于图 2-3 (a), (b) 图。

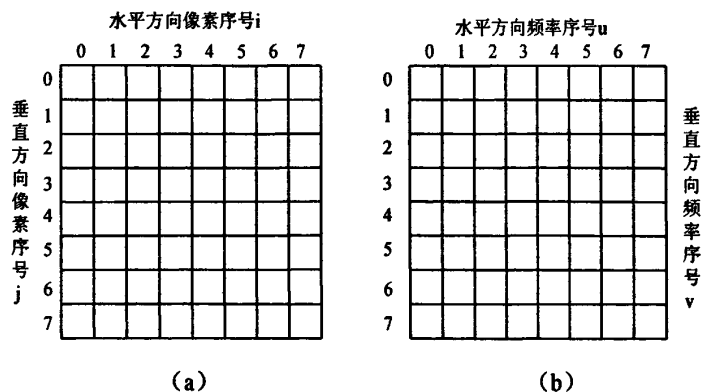


图2-3  $8 \times 8$  像素值矩阵和  $8 \times 8$  系数矩阵

$8 \times 8$  的二维 DCT 公式及 IDCT 公式<sup>[10]</sup>:

$$F(u, v) = \frac{1}{4} C(u) C(v) \sum_{i=0}^7 \sum_{j=0}^7 f(i, j) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} \quad (2-4)$$

$$f(i, j) = \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 C(u)C(v)F(u, v) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} \quad (2-5)$$

其中:  $C(u) = \frac{1}{\sqrt{2}}, u=0$        $C(u) = 1, u \neq 0$

$C(v) = \frac{1}{\sqrt{2}}, v=0$        $C(v) = 1, v \neq 0$

$f(i, j)$ : 输入(或输出)像素值,  $-128 \leq f(i, j) \leq 127$

$F(u, v)$ : DCT 系数,  $-1023 \leq f(i, j) \leq 1023$

### 2.2.1.3 一维 DCT 的快速算法

式(2-5)给出了  $8 \times 8$  的二维 IDCT 公式, 而一维的 8 点 IDCT 公式为:

$$x(n) = C \cdot a_n \cdot \sum_{k=0}^7 y(k) \cos \frac{(2n+1)k\pi}{16} \quad n=0, 1, \dots, 7 \quad (2-6)$$

其中,  $C = \sqrt{2}, a_0 = \frac{1}{\sqrt{2}}, a_n = 1, 2, \dots, 7$ 。

为把二维 IDCT 转化为一维 IDCT, 可以对式(2-5)变形, 即:

$$\begin{aligned} f(i, j) &= \frac{1}{4} \sum_{u=0}^7 \sum_{v=0}^7 C(u)C(v)F(u, v) \cos \frac{(2i+1)u\pi}{16} \cos \frac{(2j+1)v\pi}{16} \\ &= \frac{1}{8} \cdot \sqrt{2}C(v) \sum_{v=0}^7 \left[ \sqrt{2}C(u) \sum_{u=0}^7 F(u, v) \cos \frac{(2i+1)u\pi}{16} \right] \cos \frac{(2j+1)v\pi}{16} \end{aligned} \quad (2-7)$$

即  $8 \times 8$  的二维 IDCT 转化为两次 8 点的一维 IDCT 的复合运算, 先对  $8 \times 8$  矩阵的各行做一维 IDCT, 再对求得的  $8 \times 8$  IDCT 系数阵的各列求 IDCT。上式还表明, JPEG 中定义的二维 IDCT 与两次一维 IDCT 之间相差一个比例因子 0.125, 这可以通过对系数的右移来实现。

由此可见,  $8 \times 8$  的二维 IDCT 可以用 16 次一维的 8 点 DCT 来代替, 所以只要提高一维 8 点 IDCT 的速度, 也就提高了二维 IDCT 的速度。

如果直接进行 DCT 计算, 工作量很大。以  $8 \times 8$  的图像块为例, 进行 DCT 或 IDCT 需要 1024 次乘法和 896 次加法, 难以满足实时处理的需求。考虑到 DCT 具有正交性和对称性, 人们提出了一系列快速算法, 从而促进了 DCT 的实时实现。它们所需的计算量见表 2-1。(针对  $1 \times 8$  的输入信号)

用 Winograd<sup>[11]</sup>的方法证明了: 用于 8 点的一维 DCT, 理论上最少需要 11 次



乘法。Heidemmann<sup>[12]</sup>也得到了同样的结论。

表2-1 各种快速 DCT 算法

作者	Chen <sup>[13]</sup>	Wang <sup>[14]</sup>	Lee <sup>[15]</sup>	Vetterli <sup>[16]</sup>	Suehiro <sup>[17]</sup>	Hou <sup>[18]</sup>	Loeffler <sup>[19]</sup>
乘法次数	16	13	12	12	12	12	11
加法次数	26	29	29	29	29	29	29

在解码程序中实现了 Y.Arai,T.Agui,M.Nakajima<sup>[20]</sup>提出的算法,下面大致介绍一下其基本原理:

一维 DCT 可以表示为:

$$S_8(u) = \frac{C(u)}{2} \cdot \sum_{x=0}^7 f(x) \cos \frac{(2x+1)u\pi}{16} \quad (2-8)$$

二维 DCT 可以表示为:

$$S(v,u) = \frac{C(u)}{2} \frac{C(v)}{2} \cdot \sum_{y=0}^7 \sum_{x=0}^7 f(y,x) \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \quad (2-9)$$

$$\text{令 } \alpha = \frac{2x\pi u}{16}, \beta = \frac{\pi u}{16}, H = \alpha + \beta$$

$$\begin{aligned} S_8(u) &= \frac{C(u)}{2} \cdot \sum_{x=0}^7 f(x) \cos \frac{(2x+1)u\pi}{16} = \frac{C(u)}{2} \cdot \sum_{x=0}^7 f(x) \cos \left[ \frac{2x\pi u}{16} + \frac{\pi u}{16} \right] \\ &= \frac{C(u)}{2} \cdot \sum_{x=0}^7 f(x) \cos(\alpha + \beta) = \frac{C(u)}{2} \cdot \sum_{x=0}^7 f(x) \cos H \end{aligned} \quad (2-10)$$

这里对  $2 \cos H \cos \beta$  作一下变形:

$$\begin{aligned} 2 \cos H \cos \beta &= 2 \cos \beta \cos(\alpha + \beta) = 2 \cos \beta (\cos \alpha \cos \beta - \sin \alpha \sin \beta) \\ &= 2 \cos \alpha \cos^2 \beta - 2 \sin \beta \cos \beta \sin \alpha \\ &= \cos \alpha \cos^2 \beta + \cos \alpha \cos^2 \beta - \sin(2\beta) \sin \alpha \\ &= \cos \alpha + \cos \alpha \cos(2\beta) - \sin(2\beta) \sin \alpha = \cos \alpha + \cos(\alpha + 2\beta) \\ &= \cos \frac{2x\pi u}{16} + \cos \left[ \frac{2x\pi u}{16} + \frac{2\pi u}{16} \right] \\ &= \cos \frac{2x\pi u}{16} + \cos \left[ \frac{32\pi u - 2x\pi u - 2\pi u}{16} \right] \end{aligned}$$

$$= \cos \frac{2x\pi u}{16} + \cos \frac{30\pi u - 2x\pi u}{16}$$

$$\text{即 } 2\cos H \cos \frac{\pi u}{16} = \cos \frac{2x\pi u}{16} + \cos \frac{2(15-x)\pi u}{16}$$

由上面的推导可得出:

$$2\cos \frac{\pi u}{16} \cdot S_8(u) = \frac{C(u)}{2} \sum_{x=0}^7 f(x) \cdot 2\cos H \cos \frac{\pi u}{16} \quad (2-11)$$

继而有

$$\frac{4}{C(u)} \cos \frac{\pi u}{16} \cdot S_8(u) = \sum_{x=0}^7 \left[ \cos \frac{2x\pi u}{16} + \cos \frac{2(15-x)\pi u}{16} \right] \quad (2-12)$$

定义如下函数:

$$f(k) = \begin{cases} f(x) & \forall k < 8 \\ f(15-k) & \forall k \geq 8 \end{cases}$$

于是可得到

$$\frac{4}{C(u)} \cos \frac{\pi u}{16} \cdot S_8(u) = \sum_{k=0}^{15} f(k) \cos \frac{2k\pi u}{16} = \text{Re} \left\{ \sum_{k=0}^{15} f(k) e^{-i \frac{2k\pi u}{16}} \right\} \quad (2-13)$$

其中  $i = \sqrt{-1}$ 。

16 点离散傅立叶变换的定义为:

$$F_{16}(u) = \sum_{k=0}^{15} f(k) e^{-i \frac{2k\pi u}{16}} \quad (2-14)$$

由式(2-13)和离散傅立叶变换定义可得到

$$\frac{4}{C(u)} \cos \frac{\pi u}{16} \cdot S_8(u) = \text{Re} \{ F_{16}(u) \} \quad (2-15)$$

从式(2-15)可以得到启发,DCT 乘上一个常数  $\frac{4}{C(u)} \cos \frac{\pi u}{16}$  就可以得到 16 点 DFT 的

实部, 计算 IDCT 可以转化为求 IDFT。基于这样的考虑是因为, 在 IDCT 之前是反量化运算, 所以我们可以将这个常数相乘与反量化运算合并, 从而节省了乘法运算量。IDFT 运算有 16 个值, 这里实际上只需要 8 个值, 这里我们舍弃了从  $k=8 \sim 15$  之间的值。下面我们讨论如何实现 IDFT 快速算法:

定义一行 IDFT 值为矩阵:

$$X_M = (f(0) \ f(1) \ f(2) \ f(3) \ f(4) \ f(5) \ f(6) \ f(7))$$

定义矩阵  $F_M$  为:

$$F_M = \left( \frac{1}{16}F(0) \quad \frac{1}{8}F(1) \quad \frac{1}{8}F(2) \quad \frac{1}{8}F(3) \quad \frac{1}{8}F(4) \quad \frac{1}{8}F(5) \quad \frac{1}{8}F(6) \quad \frac{1}{8}F(7) \right)$$

由函数  $P(a,b) = \cos \frac{2\pi a}{16} + \cos \frac{2\pi b}{16}$ , 我们得到矩阵  $T_M$

$$\begin{pmatrix} \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} & \frac{P(0,0)}{2} \\ P(0,15) & P(1,14) & P(2,13) & P(3,12) & P(4,11) & P(5,10) & P(6,9) & P(7,8) \\ P(0,30) & P(2,28) & P(4,26) & P(6,24) & P(8,22) & P(10,20) & P(12,18) & P(14,16) \\ P(0,45) & P(3,42) & P(6,39) & P(9,36) & P(12,33) & P(15,30) & P(18,27) & P(21,24) \\ P(0,60) & P(4,56) & P(8,52) & P(12,48) & P(16,44) & P(20,40) & P(24,36) & P(28,32) \\ P(0,75) & P(5,70) & P(10,65) & P(15,60) & P(20,55) & P(25,50) & P(30,45) & P(35,40) \\ P(0,90) & P(6,84) & P(12,78) & P(18,72) & P(24,66) & P(30,60) & P(36,54) & P(42,48) \\ P(0,105) & P(7,98) & P(14,91) & P(21,84) & P(28,77) & P(35,70) & P(42,63) & P(49,56) \end{pmatrix}$$

根据式(2-12)可得到  $F_M = X_M \times \left( \frac{1}{8} * T_M \right)$

定义以下三个常数

$$K2 = \cos \frac{\pi}{8}; K4 = \cos \frac{2\pi}{8} = \frac{\sqrt{2}}{2}; K6 = \cos \frac{3\pi}{8} = \sin \frac{\pi}{8}.$$

利用 K2,K4,K6 对矩阵  $T_M$  变形为:

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1+K2 & K2+K4 & K4+K6 & K6 & -K6 & -K6-K4 & -K4-K2 & -K2-1 \\ 1+K4 & K4 & -K4 & -K4-1 & -1-K4 & -K4 & K4 & K4+1 \\ 1+K6 & K6-K4 & -K4-K2 & -K2 & K2 & K2+K4 & K4-K6 & -K6-1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1-K6 & -K6-K4 & -K4+K2 & K2 & K2 & -K2+K4 & K4+K6 & K6-1 \\ 1-K4 & -K4 & K4 & K4-1 & -1+K4 & K4 & -K4 & -K4+1 \\ 1-K2 & -K2+K4 & K4-K6 & -K6 & K6 & K6-K4 & -K4+K2 & K2-1 \end{pmatrix}$$

对  $T_M$  求逆可得

$$X_M = F_M \times (8 * T_M^{-1}) = F_M \times L; (L = 8 * T_M^{-1})$$

$$\text{令 } C2 = 2 \cos \frac{\pi}{8}; C4 = 2 \cos \frac{2\pi}{8} = \sqrt{2}; C6 = 2 \cos \frac{3\pi}{8} = 2 \sin \frac{\pi}{8}$$

矩阵  $L$  可表示成

$$L = B_1 \times M \times A_1 \times A_2 \times A_3$$

$$\text{其中 } B_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 \end{pmatrix}, A_1 = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{pmatrix}$$

$$A_2 = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}, A_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & C4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -C2 & 0 & -C6 & 0 \\ 0 & 0 & 0 & 0 & 0 & C4 & 0 & 0 \\ 0 & 0 & 0 & 0 & -C6 & 0 & C2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

于是 IDFT 运算可以表示成下面五步:

$$\begin{aligned} (1) \quad & a_0 = \frac{1}{16}F_{16}(0), a_1 = \frac{1}{8}F_{16}(4), a_2 = \frac{1}{8}F_{16}(2) - \frac{1}{8}F_{16}(6), a_3 = \frac{1}{8}F_{16}(2) + \frac{1}{8}F_{16}(6) \\ & a_4 = \frac{1}{8}F_{16}(5) - \frac{1}{8}F_{16}(3), a_5 = \frac{1}{8}F_{16}(1) + \frac{1}{8}F_{16}(7), a_6 = \frac{1}{8}F_{16}(3) + \frac{1}{8}F_{16}(5) \\ & a_7 = a_5 - a_6, a_8 = \frac{1}{8}F_{16}(1) - \frac{1}{8}F_{16}(7), a_9 = a_5 + a_6 \end{aligned}$$

$$(2) \quad b_0 = a_2 * h_4, b_1 = -(a_4 * h_2 + a_8 * h_6), b_2 = a_7 * h_4, b_3 = -a_4 * h_6 + a_8 * h_2$$

$$(3) \quad c_0 = b_3 - a_9, c_1 = c_0 - b_2, c_2 = a_0 - a_1, c_3 = b_0 - a_3, c_4 = a_0 + a_1$$

$$(4) \quad d_0 = c_2 + c_3, d_1 = c_4 + a_3, d_2 = c_2 - c_3, d_3 = c_4 - a_3, d_4 = b_1 - c_1$$

$$(5) \quad f(0) = a_7 + d_1, f(1) = d_0 + c_0, f(2) = d_2 - c_1, f(3) = d_3 - d_4$$

$$f(4) = d_4 + d_3, f(5) = d_2 + c_1, f(6) = d_0 - c_0, f(7) = d_1 - a_7$$

从步骤(1)、(2)、(3)、(4)、(5)可知, 完成 IDFT 运算只需要 5 次乘法, 29 次加法。

步骤(6)可合并到反量化运算中, 因此完成 IDCT 运算只需要 5 次乘法, 29 次加法。

$$(6) \quad F_{16}(0) = 4\sqrt{2} * S_8(0), F_{16}(1) = 4 \cos \frac{\pi}{16} * S_8(1), F_{16}(2) = 4 \cos \frac{2\pi}{16} * S_8(2)$$

$$F_{16}(3) = 4 \cos \frac{3\pi}{16} * S_8(3), F_{16}(4) = 4 \cos \frac{4\pi}{16} * S_8(4), F_{16}(5) = 4 \cos \frac{5\pi}{16} * S_8(5)$$

$$F_{16}(6) = 4 \cos \frac{6\pi}{16} * S_8(6), F_{16}(7) = 4 \cos \frac{7\pi}{16} * S_8(7)$$

### 2.2.2 DCT 系数的量化及排序

在 JPEG 中, 使用的是线性均匀量化器。若  $F(u,v)$  代表 DCT 系数,  $C(u,v)$  代表量化编码输出,  $Q(u,v)$  代表对应的量化阶距, 则  $F(u,v)$  与  $C(u,v)$  的数学关系为:

$$C(u,v) = \left\lceil F(u,v) + \frac{1}{2}Q(u,v) \right\rceil / Q(u,v) \quad F(u,v) \geq 0 \quad (2-16)$$

$$C(u,v) = \left\lfloor F(u,v) - \frac{1}{2}Q(u,v) \right\rfloor / Q(u,v) \quad F(u,v) < 0 \quad (2-17)$$

解码端有一个与量化器相反的过程, 是线性译码。若用  $F'(u,v)$  代表译码输出, 则译码过程可表示为:

$$F'(u,v) = C(u,v) * Q(u,v) \quad (2-18)$$

DCT 系数的量化过程带来了量化噪声, 而在接收端解码时并不能去除这一噪声, 考虑到哈夫曼编码是无损的, 可以得出如下结论: JPEG 编码产生的失真主要来源于 DCT 系数的量化噪声。

JPEG 推荐了一套量化矩阵, 它们是从广泛的实验中, 得到人眼对各种空间频

率的不同灵敏度之后得出的。在量化过程中，对人眼灵敏度较高的空间频率采用细量化，减少信息损失，因此使用较小量化步长，而对灵敏度较低的空间频率采用粗量化，提高压缩比率，使用较大量化步长。实施者如果想在频谱不同的区域上达到特定的效果，也可以使用自己的量化表。JPEG 使用的颜色空间是 YUV 格式，Y 代表亮度信息，U、V 代表色差信息，对人眼来说其重要性并不相同，所以 JPEG 针对亮度分量和色差分量推荐了两张不同的量化表 2-2。其中(a)为亮度量化矩阵，(b)为色度量化矩阵。

表2-2 亮度和色度量化矩阵

(a)								(b)							
16	11	10	16	24	40	51	61	18	21	26	66	99	99	99	99
12	12	14	19	26	58	60	55	24	26	56	99	99	99	99	99
14	13	16	24	40	57	69	56	47	66	99	99	99	99	99	99
14	17	22	29	51	87	80	62	99	99	99	99	99	99	99	99
18	22	37	56	68	109	103	77	99	99	99	99	99	99	99	99
24	35	55	64	81	104	113	92	99	99	99	99	99	99	99	99
49	64	78	87	103	121	120	101	99	99	99	99	99	99	99	99
72	92	95	98	112	100	103	99	99	99	99	99	99	99	99	99

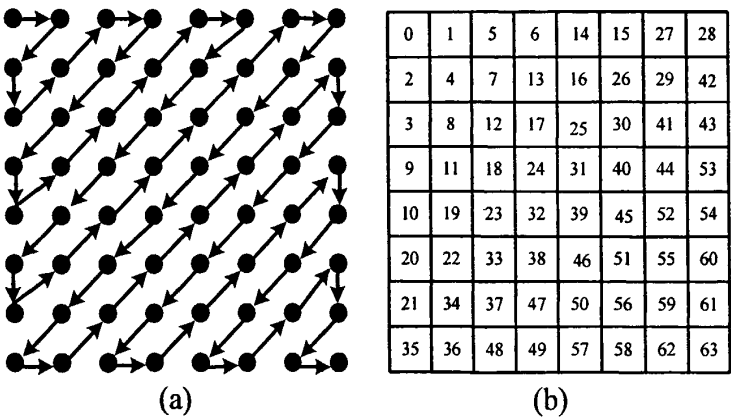


图2-4 DCT 系数的“Z”型排列和 DCT 系数序号

$8 \times 8$  的 DCT 系数矩阵可以按频率由低到高的顺序排列成一个长度为 64 的 DCT 系数向量。考虑到水平方向和垂直方向二维的频率变化,排列顺序呈“Z”字形。图 2-4(a)说明了这种排列顺序。图 2-4(b)中的  $8 \times 8$  的矩阵给出了  $8 \times 8$  DCT 系数矩阵中的各个元素在一维系数向量中的编码序号。

## 2.2.3 熵编码

### 2.2.3.1 行程编码

经过 DCT 变换后,低频分量集中在左上角,其中  $F(0, 0)$ (即第一行第一列元素)代表了直流(DC)系数,即  $8 \times 8$  子块的平均值,要对它单独编码。由于两个相邻的  $8 \times 8$  子块的 DC 系数相差很小,所以对它们采用差分编码(DPCM),可以提高压缩比,也就是说对相邻的子块 DC 系数的差值进行编码。 $8 \times 8$  的其它 63 个元素是交流(AC)系数,采用行程编码。

这 63 个 AC 系数行程编码的码字用两个字节表示,见图 2-5。

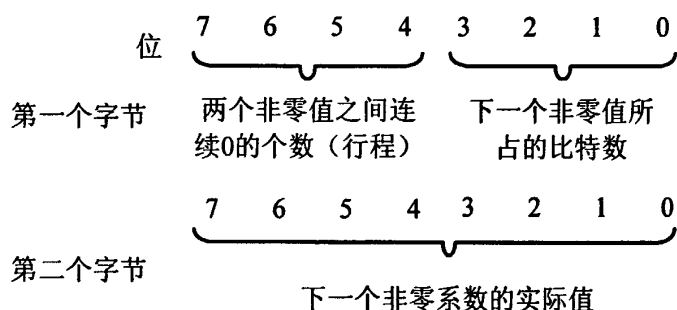


图2-5 行程编码

### 2.2.3.2 哈夫曼编码

上面,我们得到了 DC 码字和 AC 行程码字。为了进一步提高压缩比,需要对其再进行 Huffman 编码,之前我们先介绍两个概念。

熵编码的中间格式表示:对于 AC 系数,用两个符号表示。符号 1 为行程和尺寸,即上面的(RunLength, Size)。(0, 0)和(15, 0)是两个比较特殊的情况。(0, 0)表示块结束标志(EOB), (15, 0)表示 ZRL,当行程长度超过 15 时,用增加 ZRL 的个数来解决,所以最多有三个 ZRL( $3 \times 16 + 15 = 63$ )。符号 2 为幅度值(Amplitude)。

熵编码：对于 AC 系数，符号 1 和符号 2 分别进行编码。零行程长度超过 15 时，有一个符号(15, 0)，块结束时只有一个符号(0, 0)。对符号 1 进行哈夫曼编码(亮度, 色差的哈夫曼码表不同)。对符号 2 进行变长整数 VLI 编码。举例来说：Size=5 时，Amplitude 的范围是-31~-16，及 16~31，对绝对值相同，符号相反的码字之间为反码关系。所以 AC 系数为 30 的码字为 11010，31 的码字为 11011，-30 的码字为 00101，-31 的码字为 00100。符号 2 的码字紧接于符号 1 的码字之后。对于 DC 系数，Y 和 UV 的哈夫曼码表也不同。

下面我们根据例子来说明熵编码的步骤，下面为 8×8 的亮度(Y)图象子块经过量化后的系数。

13	0	-2	0	0	0	0	0
-4	-2	0	0	0	0	0	0
-3	-2	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

可见量化后只有左上角的几个点(低频分量)不为零，这样采用行程编码就很有效。

第一步，熵编码的中间格式表示：先看 DC 系数。假设前一个 8×8 子块 DC 系数的量化值为 10，则本块 DC 系数与它的差为 2，根据下列对应关系，见表 2-3：

查表得 Size=2，Amplitude=2，所以 DC 中间格式为(2)(2)。

下面对 AC 系数编码。经过 Zig-Zag 扫描后，遇到的第一个非零系数为-4，其中遇到零的个数为 1(即 RunLength)，根据 AC 系数表（见表 2-4）：

查表得 Size=3，RunLength=1,Amplitude=-4，所以 AC 中间格式为(1,3)(-4)。其余点类似,可以求得这个 8×8 子块熵编码的中间格式为(2)(2),(1,3)(-4), (0,2)(-3), (0,2)(-2), (0,2)(-2),(2,2)(-2),(EOB)(0,0)。

第二步，熵编码：

对于(2)(2)：2 查 DC 亮度 Huffman 表（表 2-5）得到 011，2 经过 VLI 编码为



表2-3 DC 系数表

Size	Amplitude
0	0
1	-1,1
2	-3,-2,2,3
3	-7...-4,4...7
4	-15...-8,8...15
5	-31...-16,16...31
6	-63...-32,32...63
7	-127...-64,64...127
8	-255...-128,128...255
9	-511...-256,256...511
10	-1023...-512,512...1023
11	-2047...1024,1024...2047

表2-4 AC 系数表

Size	Amplitude
0	0
1	-1,1
2	-3,-2,2,3
3	-7...-4,4...7
4	-15...-8,8...15
5	-31...-16,16...31
6	-63...-32,32...63
7	-127...-64,64...127
8	-255...-128,128...255
9	-511...-256,256...511
10	-1023...-512,512...1023

表2-5 亮度分量的直流哈夫曼码表

SSSS	码长	码字
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110

6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

10;

对于(1;3)(-4): (1,3)查 AC 亮度 Huffman 表 (表 2-6) 得到 1111001, -4 是 4 的反码, 为 100;

对于(0;2)(-3): (0,2)查 AC 亮度 Huffman 表得到 01, -3 是 3 的反码, 为 00;

.....

最后, 这一  $8 \times 8$  子块亮度信息压缩后的数据流为 01110,11110011100,0100,0101,0101,1111100101,1010。总共 42 比特, 其压缩比是  $64 \times 8 / 42 = 12.2$ 。

表2-6 亮度分量的交流分量编码

Run/Size	EHUFSI(码长)	EHUFCO(码字)
0/0(EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
...	...	...
F/0(ZRL)	11	11111111001
F/1	16	111111111110101
F/2	16	111111111110110
...	...	...
F/A	16	111111111111110

## 2.3 JPEG 中数据的亚抽样与内插

JPEG 中使用的彩色空间是 YCbCr, 因为亮度分量与色度分量的空间域上的宽度不等, 前者约为后者的 3 倍, 所以对色度分量要进行亚抽样, 以使得亮度分量与色度分量的频率具有相当的分辨率。亚抽样后的数据经压缩、解码后, 要经过内插, 提高色度分量的采样率才能输出。由于图像中的色度分量的频带约为亮度

分量的频率分辨率的 3 倍。但是 JPEG 中对 Y、Cb、Cr 分量的处理中不需要色度分量的频率分辨率有这么高，因此我们可以对色度分量进行水平方向和垂直方向上的亚抽样，去掉冗余信息，以获得尽量高的压缩比。

根据香农采样的理论可知，亚抽样后的色度分量的频带可以增加，而其分辨率将降低。

本文的 JPEG 范例中对亚抽样算法很简单，亚抽样后的样点是由它覆盖的像素点做算术平均得到的。

一般亚抽样有 h2v1 亚抽样和 h2v2 亚抽样，对于 h2v1 的亚抽样，即水平方向上采样间隔增加 1 倍，垂直方向上的采样间隔不变，则亚抽样前水平方向上连续两点的算术平均值即是亚抽样后对应的样点。

h2v2 的亚抽样，则是指水平方向、垂直方向上采样间隔都增加 1 倍，亚抽样前相邻四点的算术平均值作为亚抽样后的样点。

若亚抽样前后水平方向和垂直方向上的采样间隔都不变，则这种亚抽样可称为全抽样。若对 Y 分量全抽样，对 Cb, Cr 分量做 h2v2 亚抽，则抽样后的图像常称为 YUV411 的图像。若对 Y 分量做 h2v1 亚抽样，对 Cb, Cr 分量做 h2v2 亚抽样，则抽样后得到的常称为 YUV422 的图像。若对 Y 分量全抽样，对 Cb, Cr 分量做 h2v1 亚抽样，也得到 YUV422 的图像。

在 JPEG 的解码过程中，输入的压缩图像的色度分量一般是已经被亚抽样过的，对压缩图像解码后，必须对色度分量进行内插，提高其采样率，才能得到压缩前的原始图像的副本。

对数据内插，提高采样率，实际上就是使其频带变窄，而频率分辨率提高的一个过程。所以，亚抽样与内插是两个相反的过程。

对 h2v2 这种亚抽样后的数据进行内插是将水平方向和垂直方向上的采样率都提高一倍。也就是说，内插前的一个点对应于内插后的相邻的四个点。

## 第三章 JPEG 编解码算法实现的软硬件平台

本项目应飞思卡尔公司的要求，采用其微处理器 MCF5329 为硬件开发平台。本章主要介绍本文开发 JPEG 编解码算法所使用到的软、硬件开发平台。

### 3.1 硬件开发平台 ColdFire MCF5329 概述

ColdFire 是 Freescale(原 Motorola 公司半导体产品部)公司在 M68K 基础上开发的微处理芯片。ColdFire 系列芯片不仅有片内 Cache、MAC、SRAM 及 SDRAM 控制器等微处理器的特征，同时片内还具有各种接口模块。因此，ColdFire 系列芯片不但具有微处理器的高速性，还具有微控制器的使用方便等特征。

MCF5329 是基于 ColdFire V3 微处理器结构的 32 位微处理器，带有增强型乘法累加器 (Enhanced Multiply Accumulate Unit, EMAC) 的 32 位 ColdFire 内核能更好的满足文件管理、系统控制中的控制码和信号处理相结合的要求，其性能也更好。

MCF5329 也是一款通用的系统控制器，它具有出色的性能、更具竞争性的价格以及更快的速度。集成的外设、EMAC 能使 MCF5329 可以在某些应用中代替微控制器和 DSP。

#### 3.1.1 MCF5329 的特点

MCF5329 集成微处理器是 ColdFire V3 处理器内核，工作在 240 MHz，具有如下特性：

V3 ColdFire CPU

大于 211(Dhrystone 2.1)MIPS@240MHz

16KB 数据/指令 Cache

32KB SRAM

2MB Flash

增强型乘法累加器 (EMAC)

10/100 快速以太网控制器(FEC)

硬件加速加密模块

高集成度(PLL,看门狗)

USB2.0 设备控制器和收发器

I<sup>2</sup>C、SSI、QSPI 总线接口支持

3 个 UARTs

16 通道 DMA 控制器

4 通道 PWM 定时器

4 通道 32 位计时器

增强型 CAN2.0B 控制器

MCF5329 开发板的硬件结构如图 3-1 所示。

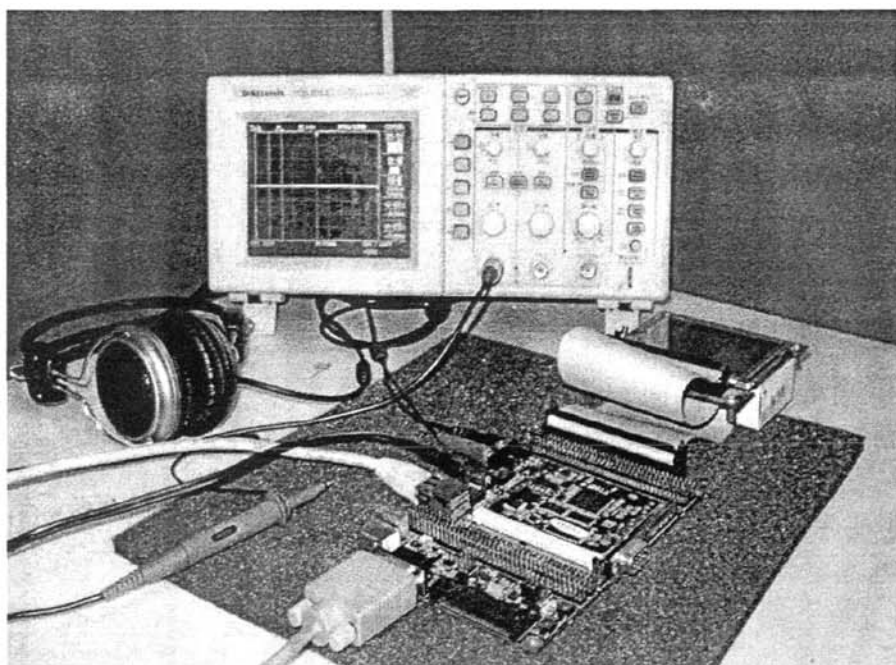


图3-1 MCF 5329 开发板实物图

### 3.1.2 MCF5329 的结构

MCF5329 的结构图<sup>[21]</sup>如图 3-2 所示。

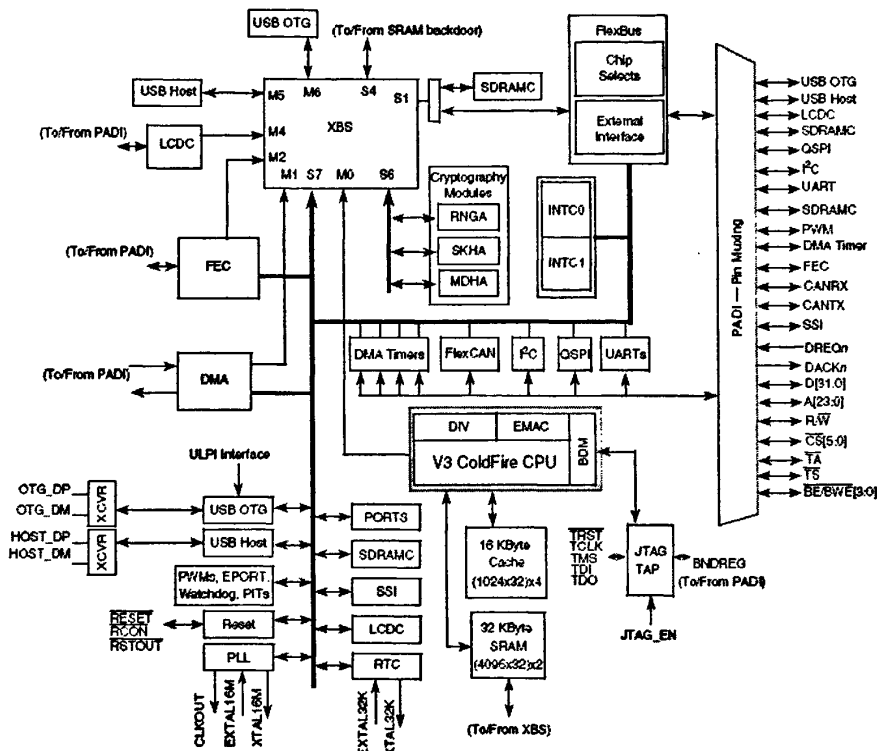


图3-2 MCF5329 结构图

### 3.1.3 ColdFire V3 内核

ColdFire MCF5329 是 V3 内核，本节重点介绍 ColdFire V3 内核的基本结构，具体有 ColdFire V3 的编程模式、流水线结构、寻址方式、乘累加单元 EMAC 以及片内存储器等。

MCF5329 是高集成度的 V3 内核产品，片内具有各种外围接口模块。V3 内核具有电路芯片面积小，代码密度高等性能。具体特性如下：

变长 RISC 指令集的 V3 微处理器内核；

2 个独立的流水线，即四级取指令流水线（IFP）和两级操作数执行流水线（OEP）；

- 32 位数据总线；
- 16 个用户可用的 32 位通用寄存器；
- 用于系统保护的管理员/普通用户模式；
- 对高级语言结构的优化。

### 3.1.3.1 分离的流水线

ColdFire MCF53xx 系列处理器基本结构如图 3-3 所示，处理器的核由两条独立流水线组成，并由一个指令缓存器完成去耦。IFP（指令提取流水）完成指令地址生成与指令提取。指令缓冲 FIFO 保留着先前取来的指令准备着送到 OEP（操作数执行流水线）去执行。而 OEP 又包含指令译码与操作数选择（DSOC）和指令执行及操作数有效地址生成（AGEX）两级流水。

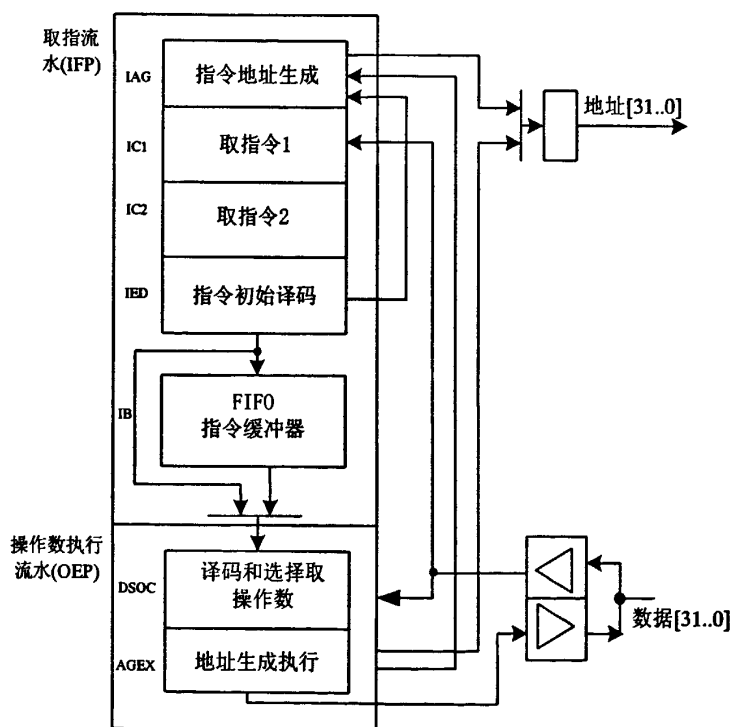


图3-3 ColdFire 处理器核的流水线结构

### 3.1.3.2 增强型硬件乘法/累加单元 (EMAC)

在 V3 核中集成的 EMAC 单元对嵌入式代码中有限的数字信号处理操作提供

硬件支持，同时支持 ColdFire 微处理器的整数乘法指令。与 V2 核 MAC 单元三级流水相比，EMAC 单元有四级执行流水线，优化了  $32 \times 32$  位乘法指令。它与 OEP 紧密地结合在一起，在单周期内通过 48 位累加器和取到的 32 位操作数执行  $32 \times 32$  位乘法。EMAC 的基本功能如图 3-4 所示。

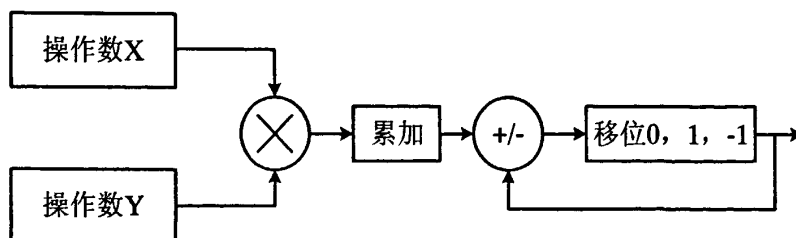


图3-4 ColdFire EMAC 功能框图

它提供了一个完整的指令集，支持有符号、无符号数和有符号定点小数的加法。MAC 可完成以下 3 种操作：①有符号和无符号整数乘法；②有符号和无符号小数的乘法累加操作；③各种寄存器操作。MAC 寄存器见图 3-5。

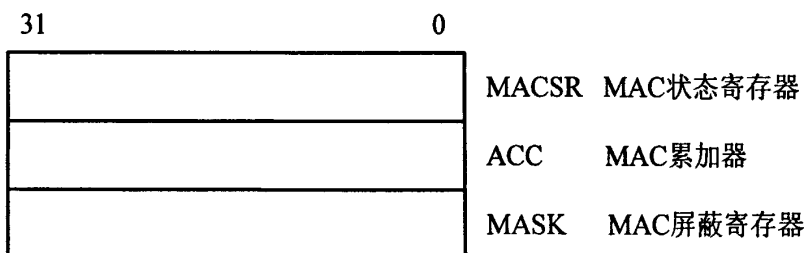


图3-5 ColdFire MAC 寄存器

### 3.1.3.3 编程模式

ColdFire 系列处理器的编程模块由两组寄存器构成：用户组和监视组<sup>[22]</sup>。运行的程序仅用到用户组中的寄存器。而系统软件运行在监视模式下时，能够访问所有的寄存器并且通过控制用户组的寄存器来完成对整个处理器系统的监视功能。V3 核处理器具有整数模式和浮点模式，各对应不同的寄存器群。其整数用户编程模块单元，在定点程序开发中发挥着及其重要的作用。

用户编程模式可以使用以下寄存器

- (1) 数据寄存器；
- (2) 地址寄存器；



(3) 32 位的程序计数器 PC;

(4) 8 位条件码寄存器。

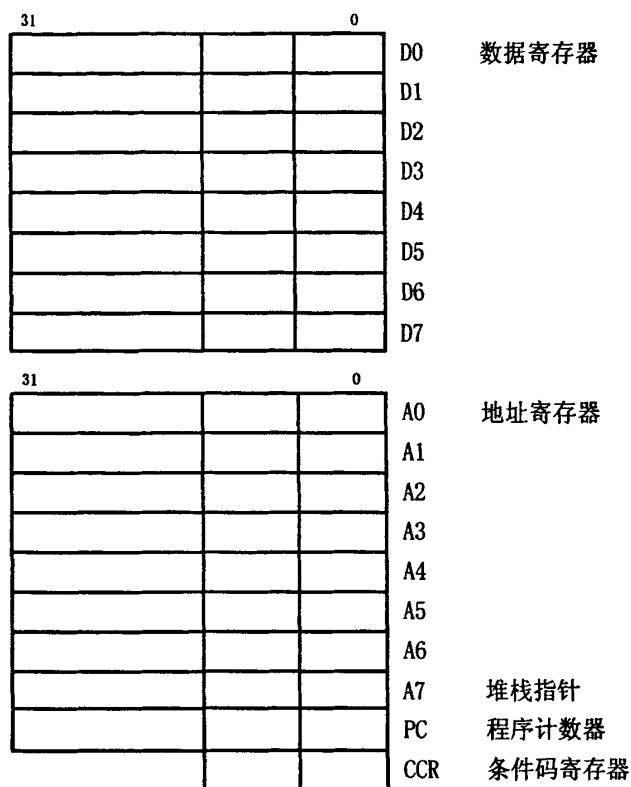


图3-6 ColdFire V3 用户编程模式寄存器

### 3.1.3.4 ColdFire 片内存储器

ColdFire V3 内核的片内存储器有 32KB SRAM 和 16KB Cache,其中 Cache 可以配置为 16KB 指令 Cache、16KB 数据 Cache 或 8KB 指令 Cache/8KB 数据 Cache。通过配置信息、取指令和读取数据通路可同时向 SRAM 和 Cache 控制器发送信息。控制器是存储器映射器件,命中/未命中由读数据访问共同决定。如果存取地址被映射到 SRAM 定义的区域(且这个区域没有被屏蔽),则把数据传回到处理器,而将 Cache 数据丢弃。如果存取地址未命中 SRAM,但是映射到 Cache 定义的区域(而且这个区域没有被屏蔽),则 Cache 把数据传回到处理器<sup>[23]</sup>。

SRAM 模块和 Cache 存取能在一个时钟周期内完成。大家知道,高速缓存的出现就是为了解决内存访问速度瓶颈的问题,当一条指令要访问某个内存地址的

时候, 如果该内存地址所处的内存块已经在高速缓存中, 那么我们就需要直接去访问内存, 而只要访问 Cache 就可以了, 这样的话, 指令执行的速度就会很快, 我们称为 Cache 命中, 反之, 如果该内存地址不在 Cache 中, 那就称之为不命中, 或叫 Cache miss, 这时就会直接访存, 下一条指令必须等这条访存指令执行完之后才能执行, 这样就产生了较多的惩罚周期。大量的访存操作直接导致了 JPEG 编解码程序在 MCF5329 嵌入式系统中执行效率低下, 所以如果可以从总体上消去一些访存操作, 可以直接减少指令条数, 也可以提升代码执行效率。

因为编解码程序中涉及到较多的数据存取, 我们将 Cache 设置成 16KB 数据 Cache。

## 3.2 软件开发平台 LTIB 工具简介

### 3.2.1 LTIB 简介

LTIB(Linux Target Image Builder, LTIB)用于建立 Linux 目标镜像。目标镜像除包含 Linux 内核外, 还包含支持各种平台的 BSPs(Board Support Packages)以及其它的一些应用包。Freescale 使用 LTIB 发布各种不同平台的 BSPs。

本课题使用的 LTIB 版本号为 m5329\_evb\_20060705, 包含以下模块:

- $\mu$ CLinux-2.6.16.1-uc0 kernel
- 板级支持包:
  - Memory devices, including Flash and NAND flash
  - TCP/IP stack & Ethernet support
  - QSPI device support
  - I<sup>2</sup>C device support
  - QSPI device support
  - Frame buffer support
- gcc 4.1,  $\mu$ Clibc 0.9.26, Binutils 2.16.91
- Debugger: 主要包含 GDB 调试工具
- 图形引擎 Microwindow 0.90
- LTIB 及  $\mu$ CLinux 预配置文件
- 说明文档

### 3.2.2 $\mu$ CLinux 操作系统

$\mu$ CLinux 是一个完全符合 CNU/GPL 公约的操作系统,完全开放源代码。 $\mu$ CLinux 中,  $\mu$  表示 Micor, C 表示 Control, 字面上的理解就是微控制领域中的 Linux 系统。它是为了支持没有内存管理单元(MMU)的 CPU 而对标准 Linux 做出的修正,因此保留了 Linux 的大多数优点:稳定、良好的移植性、优秀的网络功能、丰富的 API 等。另外, $\mu$ CLinux 内核下的二进制代码和源文件都经过了重新编写,这使得  $\mu$ CLinux 的内核同标准的 Linux 内核相比更小。概括起来,  $\mu$ CLinux 与标准 Linux 存在以下方面的差异<sup>[24]</sup>:

#### (1) 在内存保护上的差别

Linux 有内存保护而  $\mu$ CLinux 没有。对于没有内存保护的系统来说,即使一个无特权进程调用一个无效指针,也会触发一个地址错误,并潜在地引起程序崩溃,甚至导致系统的挂起。因此,对于使用  $\mu$ CLinux 的嵌入式系统而言,所运行的程序在出厂前已经固化,并对应用程序进行较完整的测试,使出现问题的概率控制在有限范围内。

#### (2) 在虚拟内存上的差别

Linux 有虚拟内存管理而  $\mu$ CLinux 没有。对于没有虚拟内存管理的系统,必须要解决下列问题:因为由内核加载的进程必须能够独立运行,并与它们在内存中的位置无关。所以,无虚拟内存管理的系统中,要么在程序被加载到 RAM 中后使程序的基地址“固定”下来,要么程序在编程中只使用相对寻址的代码。在没有虚拟内存管理的系统中,必须解决 flat 内存模型中的内存分配和释放问题。由于非常动态的内存分配会产生内存碎片,并可能耗尽系统的资源,所以,对于使用动态内存分配的那些应用程序来说,增强健壮性的一种办法使用预分配缓冲区池的办法来取代 malloc()调用。由于  $\mu$ CLinux 中不使用虚拟内存,进出内存的页面交换也没有实现,所以不能保证页面会被加载到 RAM 中的同样位置。因此在  $\mu$ CLinux 系统中,对运行的应用程序一般都限制其可分配空间不大于系统的 RAM 空间。由于缺乏内存管理的硬件单元,使得  $\mu$ CLinux 在系统接口上也会与 Linux 有些不同。像是创建子进程,  $\mu$ CLinux 要使用 vfork()创建子进程,两个进程共享他们的全部内存空间,包括堆栈。

#### (3) 在通用架构上内核的变化

在  $\mu$ Clinux 的发布版本中, `/linux/mmnommu` 目录取代了 `/linux/mm` 目录。前者就是修改后的内存管理子系统, 去除了系统对 MMU 硬件的依赖, 并在内核软件上提供了基本的内存管理函数。

本系统中所使用的  $\mu$ Clinux 版本是:  $\mu$ CLinux-2.6.16.1-uc0。

## 第四章 JPEG 编码器在 MCF5329 上的软件实现及优化

### 4.1 JPEG 算法中的数据结构及文件格式

#### 4.1.1 JPEG 的编码单元

在 JPEG 的编码或解码端，一幅图像都被看作是一个矩阵，每个矩阵元素就是其对应位置的像素值，而此矩阵内所有像素值构成了一帧。在基本系统的 JPEG 中，一幅图像只对应于一帧。

一帧图像可以在空间上分为若干部分，每一部分称为一个 scan。在基本系统中，一帧只有一个 scan。每个 scan 可以根据其所含彩色分量的数目分为两类。若只含一种彩色分量，如一个灰度图像就只含亮度分量，那么这个 scan 称为无分枝的。若此 scan 内含有一种以上的彩色分量，如一个 RGB 图像，含有红、绿、蓝三种分量，那么这个 scan 称为有分枝的。

JPEG 中对于一个 scan 的编解码是分块进行的。一个 scan 被划分为若干部分，每一部分称为一个“最小数据单元”MDU，我们在 JPEG 范例中则称其为“最小编码单元”MCU。对于无分枝的 scan 来说，一个 MCU 就是一个  $8 \times 8$  的数据块，每个块对应于图像中的一个  $8 \times 8$  的像素阵列。一个 scan 中各个块的编解码顺序是：各行的顺序是由上到下，一行内的顺序是由左到右。

对于有分枝的 scan 来说，一个 MCU 是由若干个  $8 \times 8$  的数据块组成的，scan 内的各个彩色分量分别对应一个或多个数据块。MCU 内各个数据块的顺序及数目是由 scan 内的各个彩色分量的亚抽样比率决定的。

由于一幅图像的高和宽不一定是 MCU 尺寸的整数倍，因此需要对图像的最右边一列或其最下边一行进行复制，扩展其高或宽，使得可以将整个图像划分为整数个 MCU。而在解码输出时，这些复制的行列是要被抛弃的。

#### 4.1.2 最小编码单元 MCU 的结构

MCU 的个数及 MCU 的内部组成与分量的数目及分量的抽样率有很密切的关系。若一个 scan 中定义了一种以上的分量，则这些分量的顺序及其亚抽样比率要

由帧内的分量说明来决定。这些分量说明可以确定一个 MCU 中所含各分量的  $8 \times 8$  数据块数目。一个 scan 内各 MCU 内所含数据块的总数 Nb 与此 scan 内分量数目 NS 及各分量亚抽样比率  $H_k$ 、 $V_k$  有如下关系：

$$Nb = \sum_{k=1}^{NS} H_k * V_k \quad (4-1)$$

基本系统中规定 Nb 只能取 1 到 10 的数。

由分量的亚抽样比率可确定各分量的行列数。若定义整个图像的行列数分别为 X、Y，第 K 个分量的行列数分别为  $X_k$ 、 $Y_k$ ，则  $X_k$ 、 $Y_k$  可以这样确定：

$$X_k = H_k * (X / \max(H_k)) \quad (4-2)$$

$$Y_k = V_k * (Y / \max(V_k)) \quad (4-3)$$

其中  $\max(H_k)$ 、 $\max(V_k)$  分别表示 scan 内各分量水平方向及垂直方向上亚抽样比率的最大值。

分量的亚抽样比率确定了 MCU 内各分量的数据块数目及其排列顺序。

### 4.1.3 JPEG 压缩数据格式及标记介绍

经 JPEG 算法压缩后的图像数据以按一定语法规则的压缩数据格式存放，以便在需要时经解压缩重构原来图像，或向其它系统传递数据。JPEG 压缩数据有三种不同的数据格式：

- (1) 压缩数据图像的交换格式；
- (2) 压缩数据图像的简化格式；
- (3) 表说明数据的简化格式。

其中交换格式(JPEG File Interchange Format, JFIF)<sup>[25]</sup>使用得最为广泛，它包含了图像压缩数据以及压缩时所使用的各种表说明。

JPEG 文件大体上可以分成以下两个部分：标记码(Tag)和压缩数据。标记码部分给出了 JPEG 图像的所有信息(有点类似于 BMP 中的头信息，但要复杂的多)，如图像的宽、高、Huffman 表、量化表等等。标记码的结构为：

SOI

DQT

DRI

SOF0

DHT

SOS

...

EOI

标记码可以分为两类，一类是在标记码后跟着有一段描述信息，另一类则没有。前者如 DHT、DQT 等，后者如 SOI、EOI 等。对于前类标记来说，标记后的描述信息的字节数(Field Length)用一个无符号的 16bit 整数表示，这个数即 field length，已经包括了 field length 本身的长度 2 字节。而且 field length 介于标记值和描述信息之间。下面我们把基本系统中用到的一些标记<sup>[26]</sup>作简要介绍：

SOFn(Start of Frame)标记表示一帧的开始。n 表示采用的是哪一种编码方法。

DHT(Define Huffman Table)定义一张或几张 Huffman 码表。

DRI(Define Restart Interval)表示两次复位之间的 MCU 的个数。若 DRI 一直被设置为 0，则编解码器都没有复位功能。

RSTm(Restart)表示一次复位，m 是一个模为 8 的整数，也即 m 可以取 0, 1, 2, 3, 4, 5, 6, 7。在第一次复位时，标记 RSTm 中 m 是零。

SOI(Start of Image)表示.jpg 文件的开始，其后往往跟着 SOFn 标记。

EOI(End of Image)表示 JPEG 数据流的结束。

SOS(Start of Scan)表示一个 scan 的开始，其后的信息描述的是这个 scan 内的参数。

DQT(Define Quantization Table)定义了一张或几张量化表。

DNL(Define Number of Lines)用于第一个 scan 的末尾，可以用 DNL 来重新定义帧的长度。

APPn(Application)保留给特定的应用。

DHT、DRI、DQT、和 APPn 这些标记与其信息可以出现在 SOF、SOS 标记之前。如果 DHT 和 DQT 标记及其描述信息出现在 SOI 标记后，那么它们之后可以

紧接着出现 SOI 标记。这些都是各种标记出现顺序要遵守的规则。

## 4.2 编码软件的总体结构

### 4.2.1 基于 MCF5329 的编码流程

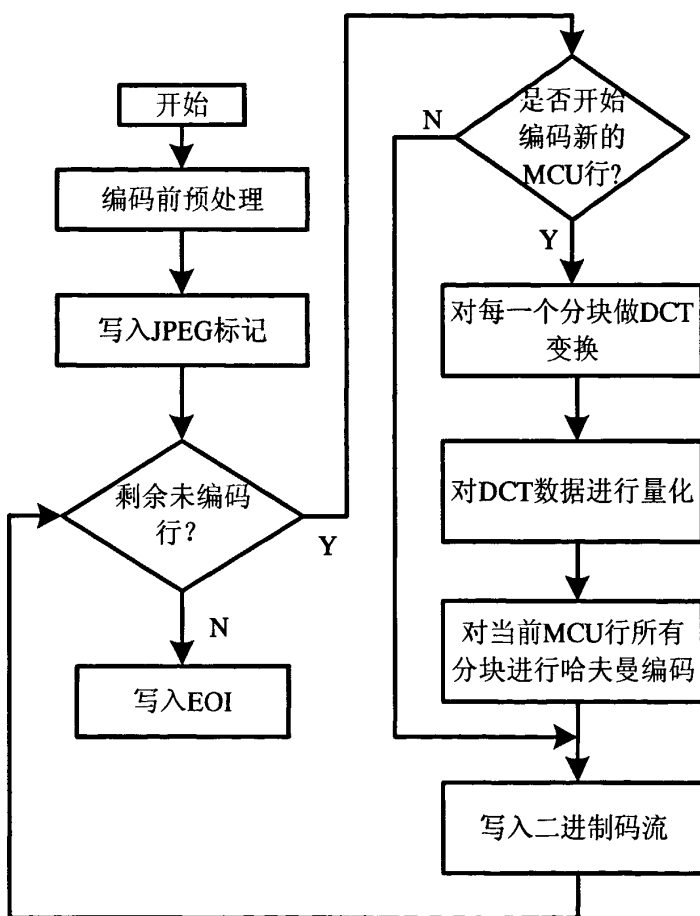


图4-1 编码流程图

JPEG 基本系统编解码器的研究主要从以下几个方面进行了考虑:首先,灵活利用 MCF5329 微处理器具有字节、半字、字、双字数据操作功能,以减少存取访问次数和数据存储空间。如压缩图像和 JPEG 图像解码后输出的数据以字节为单位,就可以减小输出缓存区间;对于 DCT、IDCT 运算,运用 ColdFire 指令进行双字操



作减少存取操作、循环次数，加快了处理速度，减少指令数量。其次，充分利用压缩图像数据的特征，优化和改进 JPEG 编解码的算法，以提高编解码速度，减少编解码过程中间数据变量缓存空间，以尽量少的 MCF5329 微处理器资源完成 JPEG 编解码功能；第三，利用 MCF5329 微处理器指令流水线特点，合理安排代码结构和指令执行顺序，提高代码效率。

从图 4-1 的压缩流程分析的结果来看，原有的程序将哈夫曼编码和 DCT 变换分别用不同的函数实现，每次都是将一个 MCU 行的数据先全部进行 DCT 变换后，再对其中每一个分块进行哈夫曼编码，这样逻辑结构清晰，很容易理解。但是在做优化的时候，提高代码执行效率才是第一原则。分析代码后发现，原来在 `my_coef_controller` 数据结构中申请了一个指针数组 `MCU_buffer[C_MAX_BLOCKS_IN_MCU]`，在编码初始化的时候为每个指针分配了一个  $64 \times 2$  字节的存储空间，而且这些存储空间是连续的，这些存储空间就是为了保存在 DCT 过程中这一个 MCU 行中所有分块的中间结果，大家知道，每一个分块都是一个  $8 \times 8$  的像素点矩阵，每个点占用 2 个字节的位置，所以每个分块的数据块的大小是  $64 \times 2$  个字节，因此每一行所有的分块所占用的数据区的大小也是不小的。在 DCT 函数 `forward_DCT()` 中有这样一个循环，顺序的先将每个 `input_buffer[i]` 指针指向的数据区的数据拷贝到一个 `workspace` 的缓冲区中，然后对这个缓冲区进行 DCT 变换，然后将结果依次保存到前面提到的数据区 `MCU_buffer[i]` 中，而在哈夫曼编码函数 `encode_mcu()` 的操作对象就是前面得到的数据区，而且也存在类似的循环。如果我们在这里将两个循环合并成一个循环，每做完一个分块的 DCT 变换后就马上做哈夫曼编码，我们就不再需要那个指针数组，只需要保留 `workspace` 缓冲区，最后直接将结果保存到 `output_buffer` 中去就可以了，这样做的话不仅省略了许多内存拷贝的工作，同时也减小了缓存大小，降低了 Data Cache 不命中的可能性。修改前后的流程见图 4-2。

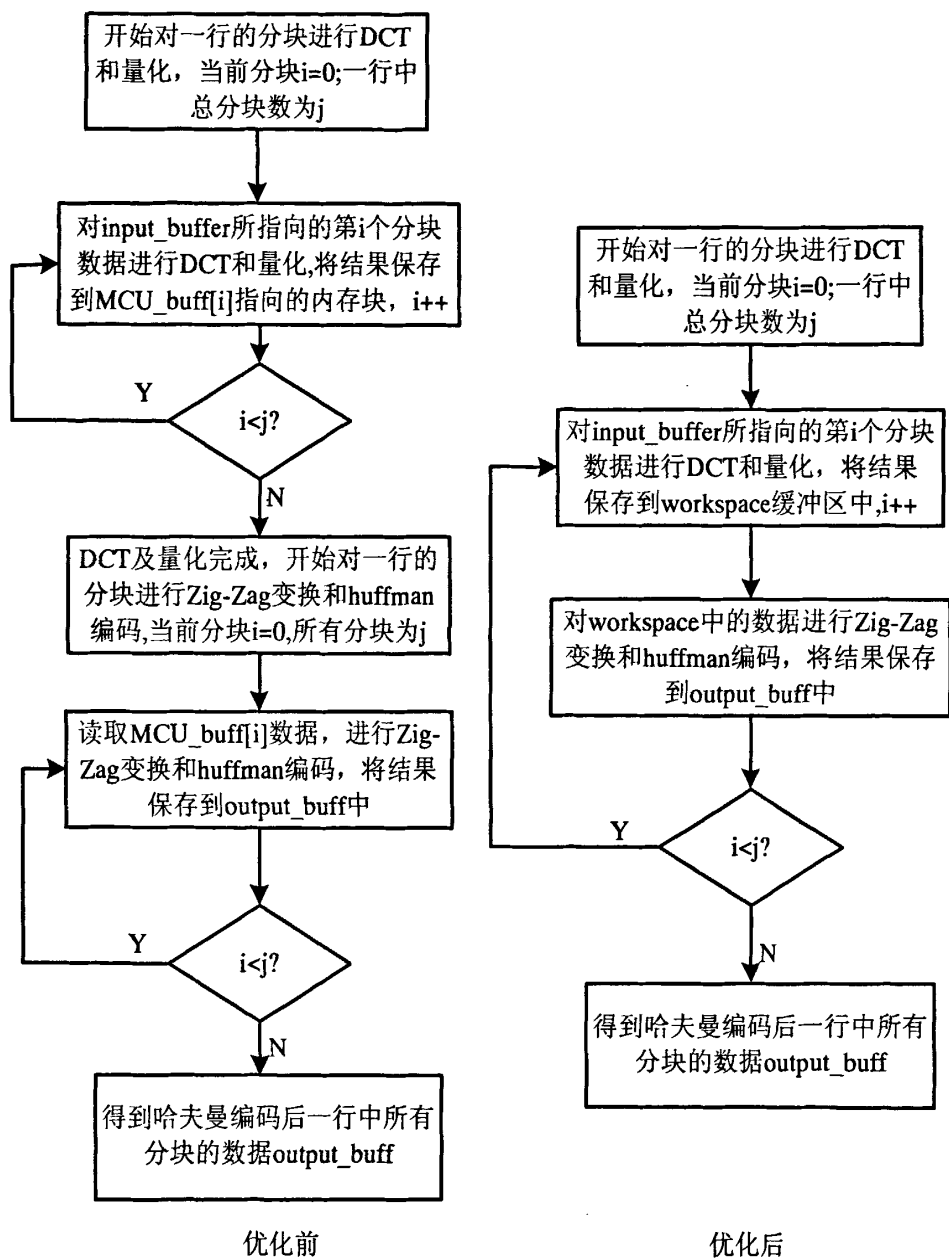


图4-2 优化前后流程图对比

### 4.2.2 编码器的默认参数设置

JPEG 编码时要设置哈夫曼码表、量化表、Q 因子、色彩空间、分量说明等码表及参数，我们预先设置了一些常见的默认参数<sup>[27]</sup>，这些参数可根据实际情况修改，以下逐一讲述其意义和值的设置。

JPEG 算法中已经给出了标准的哈夫曼码表，这种码表在实际应用中的效果令人满意，本编码器的默认设置就采用这种标准的哈夫曼码表。如果某些用户对压缩比的要求比压缩速度的要求更高，可以选择实现自适应哈夫曼码表生成的功能。

JPEG 算法中已经给出了标准的量化表，这些量化表示符合人的视觉心理的，我们这里的设置就采用这种标准的量化表。如果用户对于自己所处理的图像有一套成熟的经验，则可以自己设计量化表。

JPEG 中的 Q 因子（也称为质量系数）的质量要求。因子可以取 1 到 100 之间的整数，即  $1 \leq \text{quality\_factor} \leq 100$ ，对  $\text{quality\_factor}$  做以下映射可得到一个倍乘因子  $\text{scale\_factor}$ ：

$$\begin{aligned} \text{scale\_factor} &= 5000 / \text{quality\_factor} & \text{quality\_factor} < 50 \\ \text{scale\_factor} &= 200 - \text{quality\_factor} * 2 & \text{quality\_factor} > 100 \end{aligned} \quad (4-4)$$

$\text{scale\_factor}/100$  表示对量化表中所有的量化阶距的倍乘因子。由可以看出，Q 因子越大， $\text{scale\_factor}$  越小，量化阶距就越小，压缩后的图像质量就越高；反之，Q 因子， $\text{scale\_factor}$  越大，量化阶距越大，压缩后的图像质量越低。一般来说，如果量化阶距取 JPEG 建议中量化阶距的一半时，压缩后的图像质量是非常好的，这里我们将 Q 因子设置为 50。

本编码器要求输入图像文件是 RGB 空间的 BMP 位图，压缩在 YCbCr 空间上进行。

本编码器将 Y、Cb、Cr 三个分量初始化为 YUV411 的亚抽样比率，也即对 Y 分量做全采样，对 CbCr 分量做 h2v2 亚抽样。

### 4.3 色系变换和亚抽样算法的实现与优化

JPEG 图片使用 YCbCr 颜色模型，而非位图中所使用的 RGB 色系。JPEG 通常有两种采样方式：YUV411 和 YUV422，它们所代表的意义是 Y、Cb 和 Cr 三个

成份的数据取样比例。想要用 JPEG 基本压缩法处理全彩色图像，须先把 RGB 颜色模式图像数据，转换为 YCbCr 颜色模式的数据。通过对下列计算公式可完成数据转换：

$$\begin{aligned} Y &= 0.29990R + 0.5870G + 0.1140B \\ Cb &= 128 - 0.1687R - 0.3313G + 0.5000B \\ Cr &= 128 + 0.5000R - 0.4187G - 0.0813B \end{aligned} \quad (4-5)$$

在 4.2.2 节我们将 Y、Cb、Cr 三个分量初始化为 YUV411 的亚抽样比率，也即对 Y 分量做全采样，对 CbCr 分量做 h2v2 亚抽样。h2v2 的实现过程<sup>[28]</sup>如图 4-3 所示：

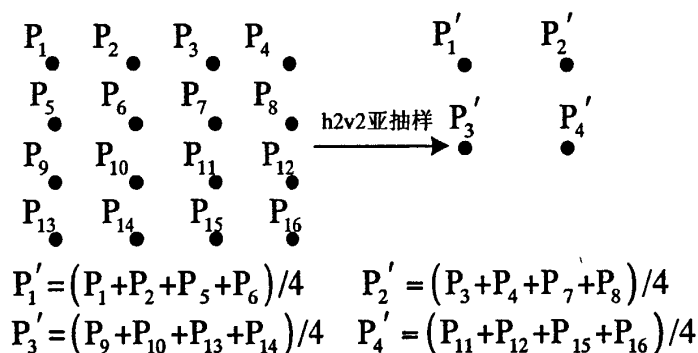


图4-3 h2v2 亚抽样

对于这两个函数，我们使用汇编语言全部改写，可以得到很高的并行性，在这个部分涉及到了许多内存读写的操作和循环跳转指令，我们通过减少寄存器相关以及减少循环次数的方法，让指令的并行性得到了很大程度的提高。

## 4.4 DCT 变换的 DSP 快速实现

DCT 算法的快慢决定了整个 JPEG 算法的速度。 $8 \times 8$  数据块的 DCT 计算有直接算法和行列法两大类，但直接算法较复杂，占用更多的 DSP 资源，因此一般使用行列法，将  $8 \times 8$  数据块的 DCT 计算转换为 16 次一维 8 点 DCT 计算，只要提高一维 DCT 的运算速度就可以提高二维 DCT 的运算速度。目前，DCT 快速算法都是针对高级语言程序设计，这些算法一般将 8 点 DCT 的计算分解为含加/减法运算和乘法运算的多级运算，如果用 DSP 实现这种多级运算，势必将时间浪费在中间数据的

存/取上,并且没有发挥 DSP 所特有的乘法累加单元的功能。

本项目中应用一种基于 DSP 乘累加单元的 DCT 快速算法<sup>[29]</sup>,DSP 一般都具有乘法累加/减(以下简称乘法累加)单元,能在单周期内完成 1 次乘法运算和 1 次累加运算。MCF 5329 具有多条乘法累加指令,其中 2 条双操作数乘法累加指令,见表 4-1 所列。

表4-1 乘累加指令

指令	表达式	周期
MAC Ry,RxSF,ACCx	$ACCx + (Ry * Rx) \{ \langle \langle \cdot \rangle \rangle \} SF \rightarrow ACC$	1
MAC Ry,Rx,<ea>y,Rw,ACCx	$ACCx + (Ry * Rx) \{ \langle \langle \cdot \rangle \rangle \} SF \rightarrow ACC, (\langle ea \rangle y) \rightarrow Rw$	3

表 4-1 中 2 条指令都能完成读出 2 个操作数、2 个操作数相乘及乘法运算结果,与 ACCx 相加后存储在 ACCx。不同的是,第二条指令能在完成乘累加的同时将在存储器中取出的数送到目的寄存器。可见这 2 条指令的功能非常强大,若能引入 DCT 计算,将大大简化程序的复杂度,并减少 DCT 运算的时间。

### (1) 八点 DCT 算法结构

将式(2-2)展开并化简与合并,得到 DCT 的输出,即:

$$Y_0 = S_0 + S_1 + S_2 + S_3$$

$$Y_4 = S_0 - S_1 - S_2 + S_3$$

$$Y_2 = S_0 \sqrt{2} \sin \frac{6\pi}{16} + S_1 \sqrt{2} \cos \frac{6\pi}{16} - S_2 \sqrt{2} \cos \frac{6\pi}{16} - S_3 \sqrt{2} \sin \frac{6\pi}{16}$$

$$Y_6 = S_0 \sqrt{2} \cos \frac{6\pi}{16} - S_1 \sqrt{2} \sin \frac{6\pi}{16} + S_2 \sqrt{2} \sin \frac{6\pi}{16} - S_3 \sqrt{2} \cos \frac{6\pi}{16}$$

$$Y_7 = -S_4 \left( \sin \frac{3\pi}{16} + \cos \frac{3\pi}{16} \right) + S_5 \left( \cos \frac{\pi}{16} + \sin \frac{\pi}{16} \right) + S_6 \left( \sin \frac{\pi}{16} - \cos \frac{\pi}{16} \right) + S_7 \left( \cos \frac{3\pi}{16} - \sin \frac{3\pi}{16} \right)$$

$$Y_3 = -S_4 \sqrt{2} \sin \frac{3\pi}{16} - S_5 \sqrt{2} \cos \frac{\pi}{16} - S_6 \sqrt{2} \sin \frac{\pi}{16} + S_7 \sqrt{2} \cos \frac{3\pi}{16}$$

$$Y_5 = S_4 \sqrt{2} \cos \frac{3\pi}{16} + S_5 \sqrt{2} \sin \frac{\pi}{16} - S_6 \sqrt{2} \cos \frac{\pi}{16} + S_7 \sqrt{2} \sin \frac{3\pi}{16}$$

$$Y_1 = S_4 \left( \cos \frac{3\pi}{16} - \sin \frac{3\pi}{16} \right) + S_5 \left( \cos \frac{\pi}{16} - \sin \frac{\pi}{16} \right) + S_6 \left( \cos \frac{\pi}{16} + \sin \frac{\pi}{16} \right) + S_7 \left( \cos \frac{3\pi}{16} + \sin \frac{3\pi}{16} \right)$$

其中,  $S_0 = x_0 + x_7, S_1 = x_1 + x_6, S_2 = x_2 + x_5, S_3 = x_3 + x_4$

$$S_4 = x_3 - x_4, S_5 = x_2 - x_5, S_6 = x_1 - x_6, S_7 = x_0 - x_7$$

从上述表达式可以看出,  $Y_0 \sim Y_7$  都是乘法累加运算, 而  $S_0 \sim S_7$  可由  $x_0 \sim x_7$  经过蝶形运算得到, 因此, 将 DCT 算法分成两级运算, 即第一级蝶形运算和第二级乘法累加运算, 减少了运算级数, 也就减少了中间数据的存、取时间。

## (2) 8 点 DCT 的 DSP 实现

第一级运算包括 4 个蝶形运算, 每个蝶形运算包括 1 次加法运算和 1 次减法运算, 可以采用 MCF5329 的双字加/减法指令。例如计算  $S_0$  :

```
move.l (a0)+,d0
```

```
mvoe.l -(a1),d1
```

```
add.l d0,d1
```

```
move.l d1,(a7)
```

$S_0$  的值从 d1 寄存器保存到账中, 这样我们将第一级所得到的  $S_0 \sim S_7$  的值都保存到栈中, 采用前递减或后递增的寻址方式为第二级运算作好准备。

第二级运算可以用乘累加的同时进行存储器间数的转移的 EMAC 指令实现。我们将第一级运算得到的  $S_4, S_5, S_6, S_7$  放到栈中, 将栈指针 a7 指向  $S_4$  的首地址, 先取出数  $S_4$  到寄存器, 进行乘累加的同时将  $S_5$  装载到寄存器, 为下一次乘累加准备数据。例如对于  $Y_1$  的实现:

```
move.l (a7)+,a0 //装载  $S_4$ , 采用后递增寻址方式将 a7 指向  $S_5$ 
```

```
moveq.l #4521, a1 //将  $(\cos \frac{3\pi}{16} - \sin \frac{3\pi}{16})$  的定点数表示装入 d1, 取 Q14
```

```
mac.l a0,d1,(a7)+,d2,acc0 //完成乘累加的同时将  $S_5$  放入 d2
```

```
moveq.l #205964,d3 //将  $(\cos \frac{\pi}{16} - \sin \frac{\pi}{16})$  的定点数表示装入 d3
```

```
mac.l d2,d3,(a7)+,d0,acc0
```

```
moveq.l #19265,d1
```

```
mac.l d0,d1,(a7)+,d2,acc0
```

```
moveq.l #4520,d3
```

```
mac.l d2,d3,(a7),d0,acc0 //acc0 中保存的是  $Y_1$  的值, 将  $S_7$  装入 d0, 计算下一
```

个值,此时 a7 可用前递减寻址方式,要特别注意 a7 所指向的地址,否则极易出错

计算出一行的值后,下一行的值可以类似地计算,这里没有用循环操作,从而可以节省很多判断和跳转指令,提高程序的效率。

采用不同方法计算 8 点 DCT 的时间,见表 4-2 所列。

表4-2 DSP 计算 8 点 DCT 时间

	Loffler 算法	DSP 直接优化算法
时钟周期数	672	240
t/μs	2.8	1.0

#### 4.5 DCT 系数量化的实现及优化

设 DCT 变换后  $8 \times 8$  系数矩阵的元素为  $D_{xy}$ , 相应的量化矩阵元素为  $Q_{xy}$ , 则量化后的系数矩阵相应元素为  $C_{xy} = \text{Round}(D_{xy} / Q_{xy})$ 。但是由于 MCF5329 是定点整数运算处理器, 不提供整数除法指令, 除法运算只能调用数学仿真功能, 而这样的整数除法往往需要约三十个时钟周期。由于  $8 \times 8$  的系数矩阵需要 64 次除法操作, 而且对每个 MCU 都进行, 故需对量化运算进行优化, 将耗时的除法改为定点乘法和移位运算, 将显著减小运算量。

浮点数需要乘一个比例因子并取整变成整型数后, 才能使用定点运算指令。在转换中必须保持系统的比例因子的一致性。因为定点运算指令认为操作数都是整数, 做加减运算不存在对阶 (即小数点位置) 问题, 只有所有数的比例因子一致才能得到正确结果。

通常用 Q 格式数表示浮点数。以 32 位定点数为例: 最高位为符号位, 如规定符号位后即是小数点位置, 则称为 Q31 格式数, 其表示的实际小数范围是  $-1 \leq x \leq 0.999999999534339$ 。设以 q 代表所定义的 Q 格式值, 则定点数  $X_q$  与实际小数 x 之间的转换关系<sup>[30]</sup>为:

$$X_q = \text{int}(x * 2^q), \quad x = X_q * 2^{-q}$$

将  $D_{xy} / Q_{xy}$  进行改写:

$$\begin{aligned}
 D_{xy} / Q_{xy} &= (D_{xy} \times SCALE) / (Q_{xy} \times SCALE) \\
 &= (D_{xy} \times (SCALE / Q_{xy})) / SCALE \\
 &= (D_{xy} \times Q'_{xy}) / SCALE
 \end{aligned}$$

这里需要合理选取 Q 格式值的大小, 即 SCALE 的值。SCALE 应取得较大, 从而提高  $SCALE / Q_{xy}$  即  $Q'_{xy}$  的精度, 因为 ColdFire 指令集一次最多只能移位 7, 我们选取 Q 值为 14, 即  $SCALE = 2^{14} = 16384$ 。在这里我们将量化和二维 DCT 转化为一维 DCT 之间相差的 8 合并进行, 即原量化表中系数乘以 8 后再求倒数, 再乘以 16384 得到定标以后的新量化表。量化完成以后所有的值要进行右移位 14 的运算, 就可以得到量化以后的实际值。

将除法运算改为定点乘法后, 对每个 MCU 的量化只需做快速的乘法(用 EMAC 单元实现, 1 个 DSP 时钟周期)和快速的符号数右移位操作(2 个 DSP 时钟周期), 这样量化运算速度提高了 8 倍多。

表4-3 定点亮度量化表

128	186	205	128	85	51	40	34
171	171	146	108	79	35	34	37
146	158	128	85	51	36	30	37
146	120	93	71	40	24	26	33
114	93	55	37	30	19	20	27
85	59	37	32	25	19	18	22
42	32	26	24	20	17	17	20
28	28	22	21	18	20	20	21

表4-4 定点色度量化表

114	85	44	21	21	21	21	21
98	79	31	62	21	21	21	21
79	37	21	21	21	21	21	21
21	21	21	21	21	21	21	21



21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21
21	21	21	21	21	21	21	21

## 4.6 哈夫曼编码的实现与优化

在 2.2.3.2 中我们大致介绍了哈夫曼编码的基本原理，直流和交流哈夫曼编码都是通过查表实现，下面分别介绍。

### 4.6.1 直流系数的哈夫曼编码

为了提高编码速度，常常定义了两张码表 XHUFISI 和 XHUFICO。这两张码表可看作是表 2-5 和色度分量的直流哈夫曼码表的扩展。利用这两张码表就可以对直流系数进行编码了。

扩展码表 XHUFISI 和 XHUFICO 的索引包含了直流分量差值所有可能取的值，其中 XHUFICO 给出了对应于每个差值的 VLC 及 VLI，XHUFISI 给出了对应于每个差值的 VLC 及 VLI 位数和。

利用扩展码表 XHUFISI 和 XHUFICO 的编码步骤是：

- (1)  $DIFF = DC - PRED$ ;
- (2)  $SIZE = XHUFISI(DIFF)$ ;
- (3)  $CODE = XHUFICO(DIFF)$ ;
- (4) 发送 SIZE 位的 CODE.

其中 DC 为直流分量，PRED 为其预测值，DIFF 表示差值信号。

因为基本系统中的输入数据精度为 8 位，直流分量的差值不会太大，所以扩展码表 XHUFISI 和 XHUFICO 的尺寸不算大，用它们进行查表编码的速度是令人满意的。

## 4.6.2 交流系数的哈夫曼编码

在 JPEG 中定义了两张长度为 256 的一维码表 EHUFFSI 和 EHUFCE, 分别表示变长码 VLC 的长度和变长码本身。

利用 EHUFFCE 和 EHUFFSI 两张码表就可以对一维系数向量 ZZ 进行 Huffman 编码了。图 4-4 是交流分量的 Huffman 编码的流程图。

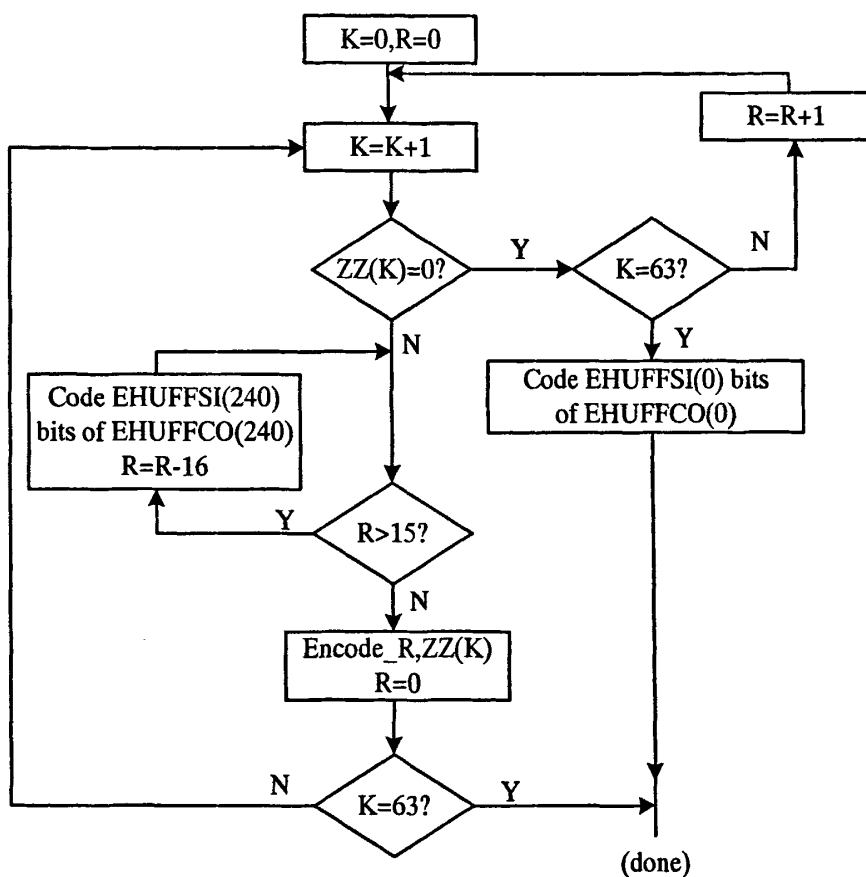


图4-4 交流系数的编码

图 4-4 中,  $K$  表示交流系数在一维系数向量  $ZZ$  中的位置,  $R$  表示连续出现零系数的个数。其中“Code EHUFFSI(240) bits of EHUFFCE(240)”是 ZRL 码字的生成, 表示连续 16 个零系数的出现。步骤“Code EHUFFSI(0) bits of EHUFFCE(0)”则是 EOB(End of Block)码字的生成, 若向量  $ZZ$  的最后一个元素不为零, 则不产生 EOB

码字。

图 4-4 中的“ENCODE\_R ZZ(K)”表示对非零系数 ZZ(K)编码，这一过程的流程图见图 4-5。

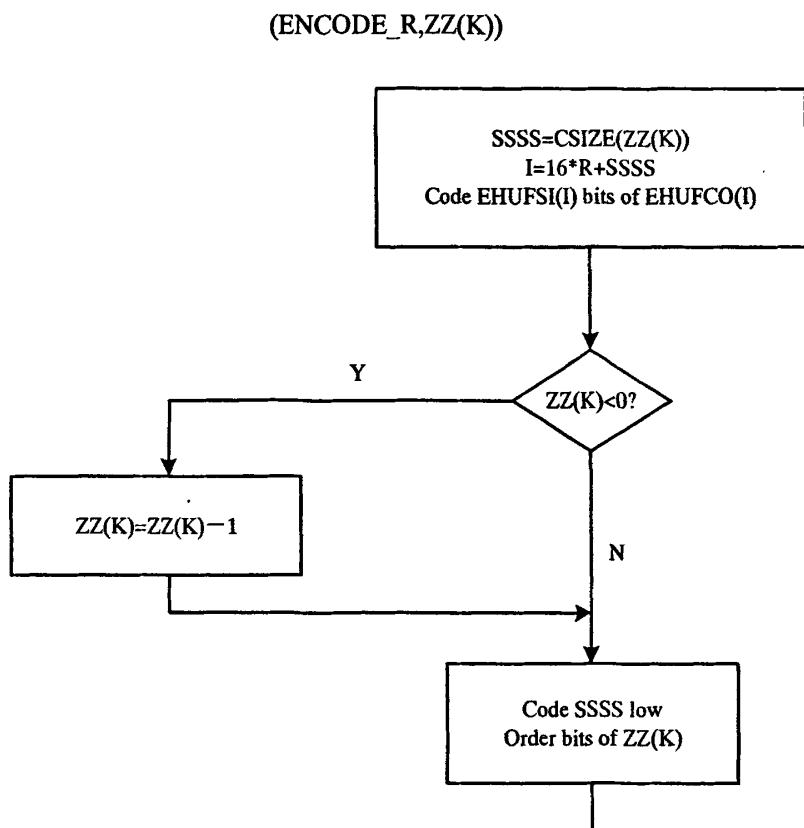


图4-5 ENCODE\_R,ZZ(K)

图 4-5 中，CSIZE()的功能是把交流系数映射成 SSSS。步骤“Code EHUFFSI(I) bits of EHUFFCO(I)”表示变长码 VLC 的生成，而“Code SSSS low order bits of ZZ(K)”即变长整数 VLI 的生成。

### 4.6.3 哈夫曼编码的优化

在哈夫曼编码时，要将量化后的 DCT 系数进行 Zig-Zag 排列，以尽可能多地 将系数 0 组合在一起进行行程编码。程序具体实现过程是定义了一个表 jpeg\_natural\_order[i]，表中定义了  $8 \times 8$  block 中 64 个系数 Zig-Zag 变换以后的位置。

每次得到一个系数前，都要查表以确定在原 block 中的位置，因此一个 block 需要查表 64 次。研究表明，DCT 实际上是空间域的低通滤波器，经过 DCT 变换及量化后左上角的低频部分将会保留，右下角的高频部分将变为 0。基于这样的考虑，我们可以将循环分成两部分，第一部分非 0 系数部分循环 15 次（一般情况下左上角非 0 系数不会超过 15），其余部分将循环展开，直接使用表中各系数的位置，省略查表步骤。具体修改如下：

```
原程序：for (k = 1; k < 64; k++) {
    if ((temp = block[jpeg_natural_order[k]]) == 0) {
        r++; }
    else {...}}
```

```
修改为：for (k = 1; k < 16; k++) {
    if ((temp = block[jpeg_natural_order[k]]) == 0) {
        r++; }
    else {...}}
    while(!(block[2]))
        r++;}
    else{...}}
    while(!(block[18]))
        r++;}
    else{...}}
    ...
```

虽然增加了一定的代码量，但经过测试证明，程序执行速度确有一定程度的提高。

## 4.7 测试及结果分析

### 4.7.1 压缩前后图像对比分析

压缩前的图片 lena.bmp 如图 4-6 所示。



图4-6 lena.bmp (512×512, 24 位色, 大小为 768KB)



图4-7 Q=100, 文件大小为 179KB, 压缩比为 4.3



图4-8 Q=50,文件大小为 24.3KB,压缩比为 31.6



图4-9 Q=20,文件大小 13.7KB, 压缩比为 56.1

图 4-7、图 4-8 和图 4-9 为设置不同 Q 值得到的压缩后的 JPEG 图片。从以上图片可以看出,质量参数从 100 到 30 变化,图片质量没有明显下降,但质量参数降到 20 以后,图像开始明显失真,出现比较明显的方块效应,这也是 JPEG 算法在图片压缩比率较高时的固有缺陷。在实际应用中一般质量参数控制在 50 左右,

能完全满足要求，既可以有较高的压缩比又能有较好的图像质量。

4.7.2 算法复杂度的测试与分析

4.7.2.1 时间复杂度

时间复杂度是指压缩一幅图片所需要的时间，对一幅 240×320 的图片进行压缩，所用时间如表 4-5 所示。

表4-5 优化前后编码时间比较

	优化前	编译器优化(O3)	手工优化
240×320 (24 位图)	0.413s	0.394s	0.227s
240×320 (灰度图)	0.333s	0.320s	0.175s

观察表 4-5 中数据，我们发现编译器对编码软件的优化的贡献很低，很难满足实际应用需求，因此我们需要对编码软件进行手工优化。在 JPEG 编码算法中，DCT 变换、量化、哈夫曼编码几个部分占了整体编码时间的 70%以上，所以对 JPEG 编码软件部分的优化我们主要集中在这几个部分，表 4-6 列出了各个模块优化以后的效果。编码速度提高率的计算公式如下：

编码速度提高率 = (优化前运行时间-优化后运行时间) / 优化前运行时间

表4-6 各模块优化后效果

优化部分	编码速度提高率
编码流程	4.7%
色系变换和亚抽样	3.5%
DCT 变换	12.8%
DCT.系数量化	22.8%
哈夫曼编码	0.3%
总计	44.1%

从表 4-6 可以看出，对色系变换、亚抽样和哈夫曼编码几个模块的优化效果不是很理想。编码的时候需要对色度分量进行色系变换和亚抽样，虽然色系变换子

程序和亚抽样程序耗时并不长，但是中间涉及到大块的数据拷贝，从而使整个编码时间明显增大。

由于哈夫曼编码模块的逻辑关系复杂，代码的前后依赖关系很大，所以要使程序并行性大幅度地提高的可能性不高；我们曾经试图将一些调用频繁的函数用汇编代码实现，或者是将有的地方用内嵌汇编替代，但是这样做的结果是，使用这些汇编代码的函数便不能作为内联函数来编译，或者说通过了编译，但实际并没有作为内联函数展开，反而增加了调用开销，导致代码数量的增加，所以对这部分的优化，我们主要致力于减少代码访存次数和提高程序结构的合理性。

#### 4.7.2.2 空间复杂度

空间复杂度是指完成 JPEG 编码算法所需要的存储空间（单位为 K 字节）。编译后的可执行代码所占存储空间如表 4-7 所示。

表4-7 优化前后编码器所需存储空间

	优化前	优化后
编码器大小 (KB)	91.8	130.2

从表 4-7 可以看出优化后代码明显增大，这是因为执行 O3 级别优化后，编译器将自动展开函数调用，执行循环展开，内联调用小功能模块，这些优化都使得 JPEG 编码器所占存储空间增大了不少。如果对编译后的程序所占存储空间有特定要求，可以调低编译器优化级别，或者手工进行优化。

#### 4.7.3 压缩比和压缩质量评价

对一幅 240×320 的 24 位位图进行压缩后，原始 bmp 格式图片大小为 224KB，压缩后的 JPEG 图片大小为 9.84KB，则压缩比计算如下：

$$\text{压缩比} = 224/9.84 = 22.8$$

对一幅 240×320 的 8 位灰度图进行压缩，原始 bmp 格式图片大小为 75.8KB，压缩后的 JPEG 图片大小为 7.8KB，则压缩比计算如下：

$$\text{压缩比} = 75.8/7.8 = 9.7$$

在前面 1.2.3.2 节介绍了 PSNR（峰值信噪比）参数是图像质量评价中常用的客观参数，其计算公式<sup>[31]</sup>为：



$$PSNR = 10 \log_{10} \left[ \frac{MNf_{\max}^2}{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [f(i, j) - f'(i, j)]^2} \right] \quad (4-6)$$

在我们的测试中， $f_{\max}=255$ 。一般说来，PSNR 值大于 30 表明与原图像的失真在可接受的范围内，表 4-8 列出了一幅  $240 \times 320$  的图片经不同压缩比压缩后，再解压缩出来的图像的 PSNR 值。

表4-8 不同压缩比率下图像的 PSNR 值

质量系数		100	75	50	30
压缩比率		6.3	18.4	22.8	33.4
文件大小(KB)		35.8	12.2	9.8	6.7
PSNR (dB)	R	38.4	35.8	32.4	27.5
	G	38.6	35.2	32.1	27.2
	B	38.9	35.1	31.9	27.4

从表 4-8 中可以看出，当质量系数在 50 以上时，PSNR 值在 30 以上，压缩出来的图像质量较好，失真在可接受范围内；当质量系数设为 30 时，PSNR 值急剧下降，失真比较严重。如果追求较高品质图像，不宜采用过高压缩比率。

## 第五章 JPEG 解码器在 MCF5329 上的软件实现及优化

### 5.1 基于 MCF5329 的 JPEG 压缩图像解码流程

JPEG 解码器大体上的程序流程如图 5-1 所示。

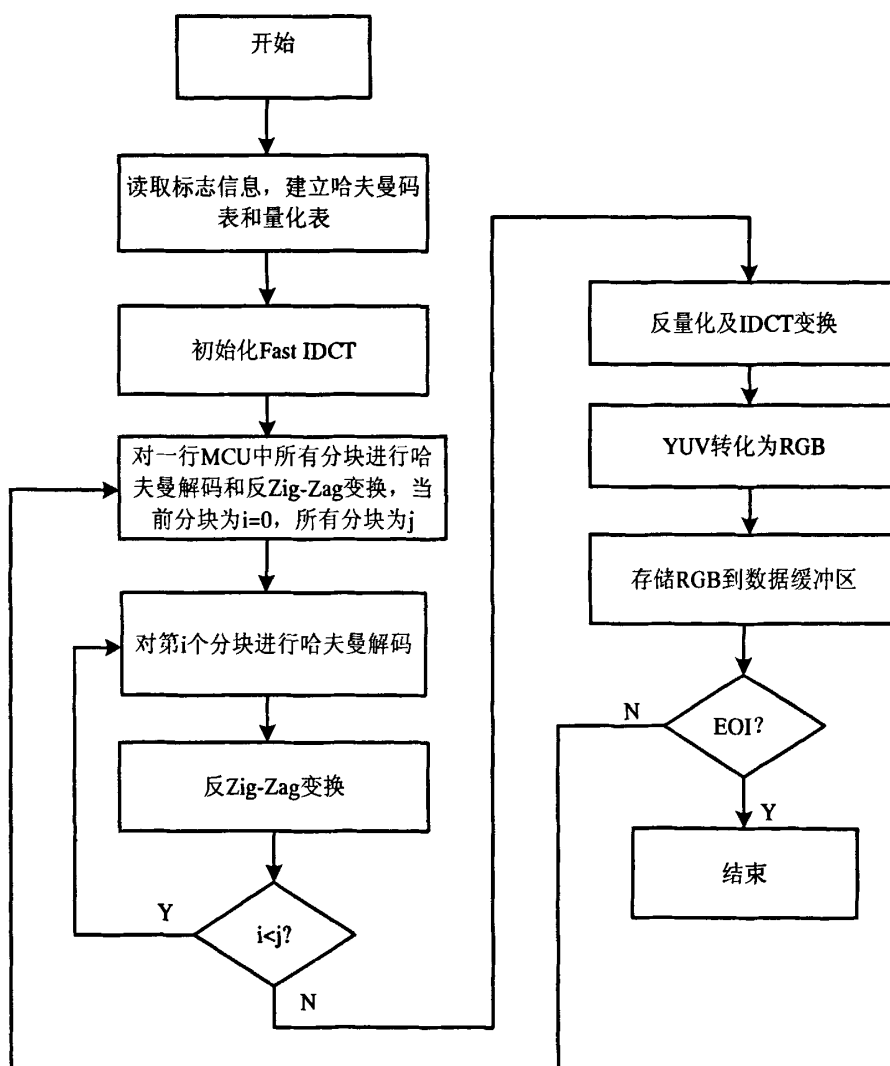


图5-1 JPEG 解码器基本流程图

在 MCF5329 上实现该流程图的时候我们做了一定的改进, 改进后的流程图如

图 5-2 所示。

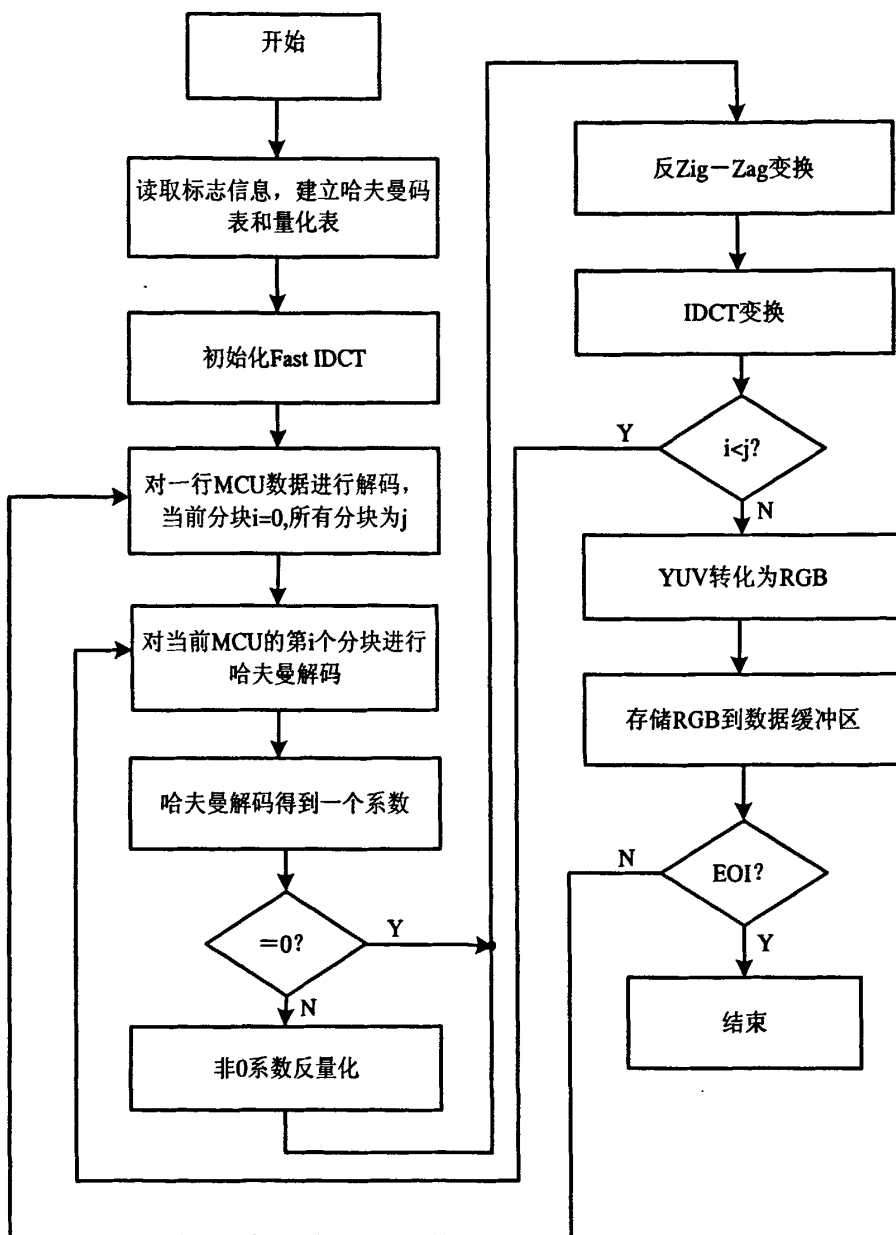


图5-2 基于 MCF5329 核微处理器解码流程图

在 JPEG 解码器基本流程图的基础上我们做了两点改动, 一点跟前面 JPEG 编码器类似, 在一行 MCU 中有多个  $8 \times 8$  数据块, 我们做完一个  $8 \times 8$  数据块的哈夫曼解码、反 Zig-Zag 变换及反量化后马上作 IDCT 变换, 这样前面分配的缓冲区

MCU\_buffer[D\_MAX\_BLOCKS\_IN\_MCU]就不需要了，这样做的话不仅省略了许多内存拷贝的工作，同时也减小了缓存大小，降低了 Data cache 不命中的可能性。

另外一点是在解码过程中，将哈夫曼解码和反量化结合起来进行处理，下一节将有专门的讨论。

## 5.2 哈夫曼解码及反量化的实现

在 Huffman 解码端接收到的是由变长码 VLC 和变长整数 VLI 组成的数据流。为了从这数据流中恢复编码前的 DCT 系数，JPEG 中定义了三张解码码表，分别介绍如下：

- MINCODE (L): 码长为 L 的最小码字
- MAXCODE (L): 码长为 L 的最大码字
- VALPTR (L): MINCODE (L) 在 HUFFVAL ( ) 中的序号

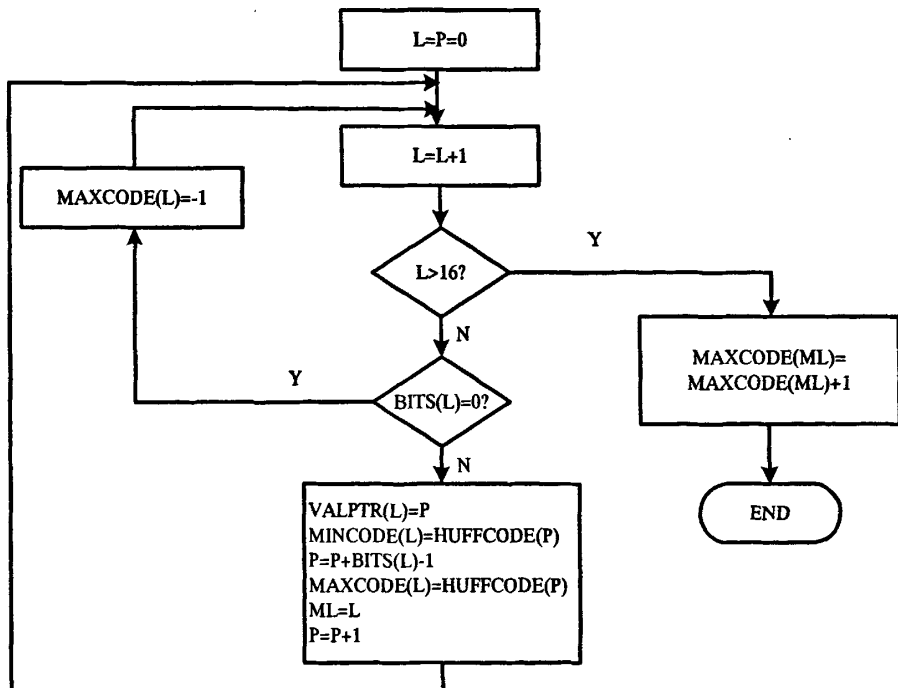


图5-3 MINCODE()、MAXCODE()、VALPTR()的生成

MINCODE ()、MAXCODE ()、VALPTR () 都是以码长为序号的长度为 16 的一维数组。MINCODE () 与 MAXCODE () 中的元素都是 16 位带符号整数, 这 3 张码表的生成见图 5-3 所示。

利用 MINCODE ()、MAXCODE ()、VALPTR () 进行 Huffman 解码的流程图如图 5-4 所示。这个过程记为 DECODE。DECODE 的输出是一个 8 位的值 VALUE。图中的“NEXTBIT”是指从数据流中读取 1 位的操作, 其流程图见图 4-6 中。NEXTBIT 这一步骤不仅读取 1 位数据, 而且能够进行填充字的删除、标记的译码等工作。如果在做 Huffman 编码时碰巧产生了一个 0xFF, 那么就用 0xFF 0x00 代替。即是说在 JPEG 图形解码时碰到 FF00 就把它当作 FF 处理。另外在 Huffman 编码区域结束时, 碰到几个 bit 没有用的时候, 应该用 1 去填充, 然后后面跟 FF。图 5-5 中的 CNT 在编译码开始或复位时都被清零。

DECODE 过程返回的是一个变长码 VLC, 它指出其后的变长整数 VLI 的位数。软件中还定义了一个读取变长整数过程 RECEIVE (S), 它返回 S 位的整数 C。RECEIVE (S) 的流程见图 5-6 (a) 中。

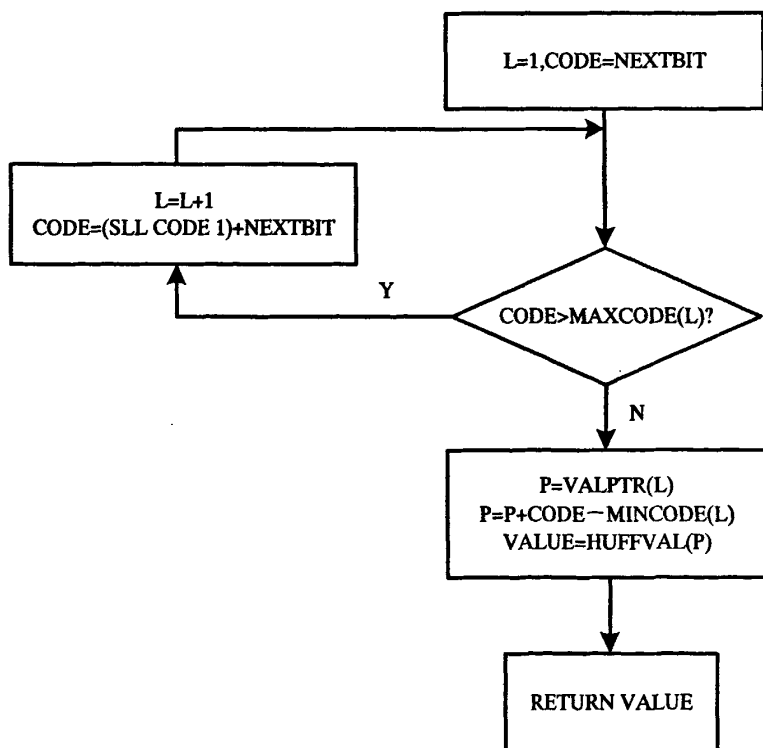


图5-4 DECODE 过程

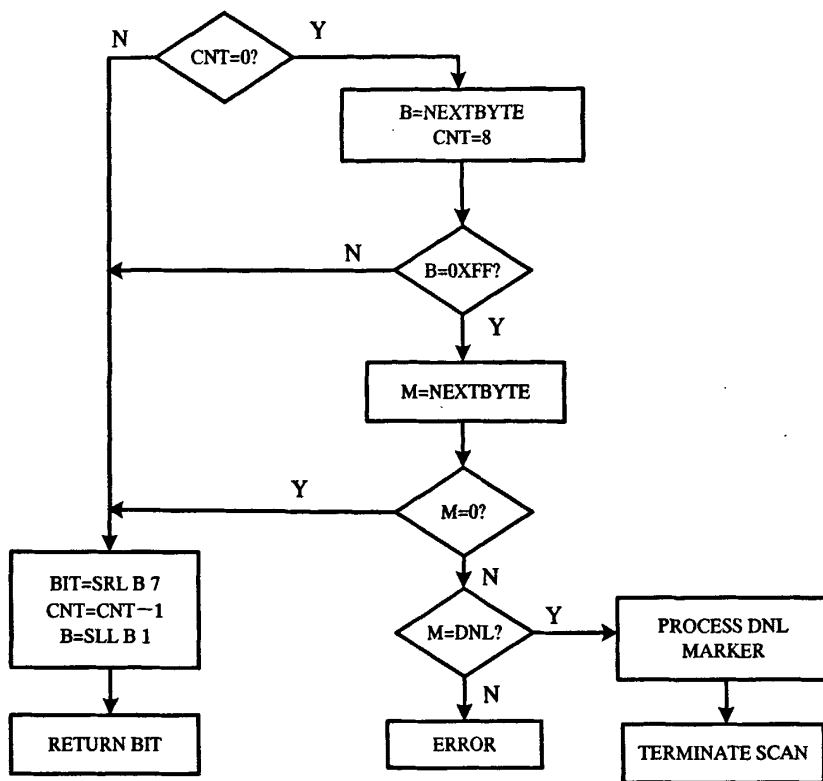


图5-5 NEXTBIT 过程

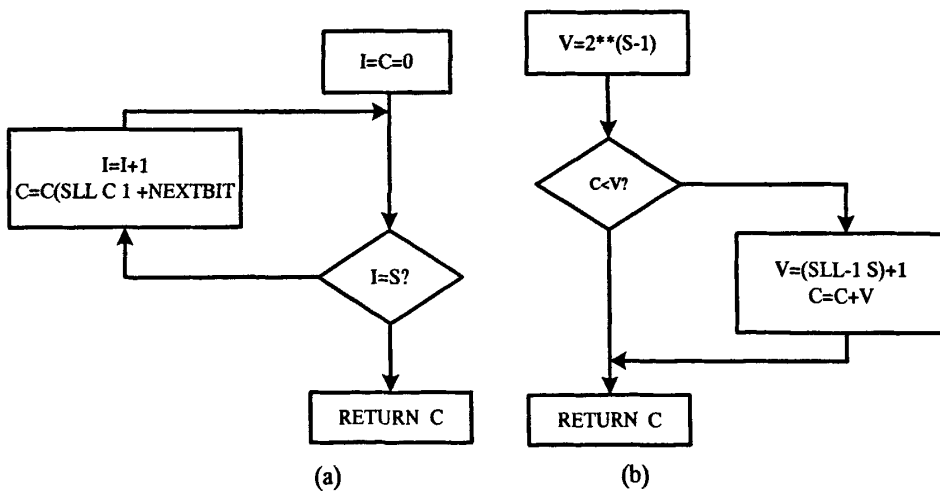


图5-6 RECEIVE(S)和 Huff\_EXTEND(C,S)

利用 DECODE ( ) 与 RECEIVE ( ) 可以对直流 DCT 系数及交流 DCT 系数的

Huffman 码解码，下面分别叙述。

## 5.2.2 直流 DCT 系数的哈夫曼解码

对直流系数的 Huffman 解码较为简单，共分为以下 4 步：

- (1)  $S = \text{DECODE}()$  ;
- (2)  $\text{DIFF} = \text{RECEIVE}(S)$  ;
- (3)  $\text{DIFF} = \text{Huff\_EXTEND}(\text{DIFF}, S)$  ;
- (4)  $\text{DC} = \text{PRED} + \text{DIFF}$ .

这里对 Huff\_EXTEND() 这一过程做如下解释。经 RECEIVE(S) 得到的差值信号 DIFF 的特点是：若 DIFF 的最高位为 1，则表示正数；若 DIFF 的最高位为零，则表示负数，要对此数加 1，并在最高位前添“1”，才能恢复出 Huffman 编码前的负数。Huff\_EXTEND() 的流程图见图 5-6(b)中。

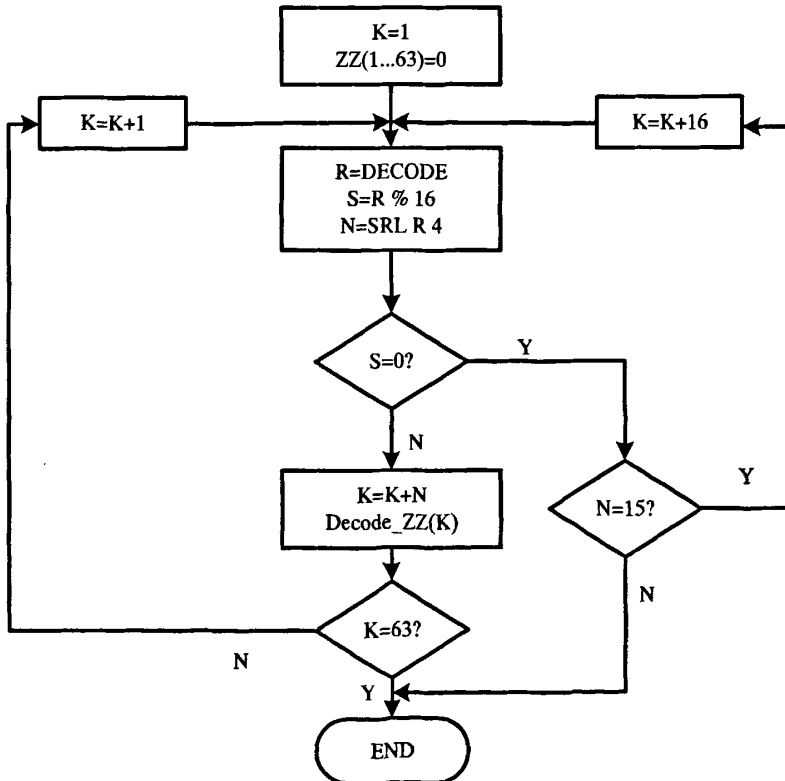


图5-7 交流系数的哈夫曼解码

### 5.2.3 交流 DCT 系数的哈夫曼解码

对交流系数的 Huffman 解码的结果是给长度为 63 的交流系数向量  $ZZ(1, 2, \dots, 63)$  的各元素都赋一个值。其解码流程见图 5-7 中。

图 5-7 中, 步骤 “Decode\_ZZ(K)” 表示对一个非零系数的幅度及符号的解码, 其流程见图 5-8 中。注意, 对零系数的解码已通过对  $ZZ(1, \dots, 63)$  的清零完成了。

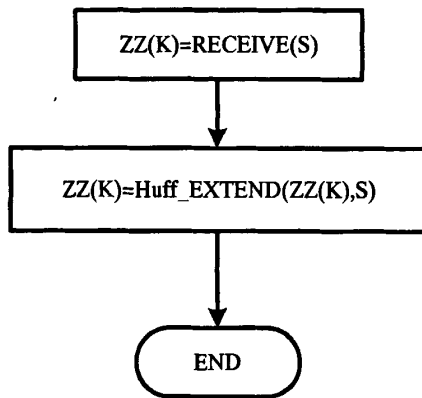


图5-8 Decode\_ZZ(K)

### 5.2.4 哈夫曼解码与反量化结合的优化

研究表明, 一个  $8 \times 8$  的图象块经过 DCT 变换后, 其低频分量都集中在左上角, 高频分量分布在右下角(DCT 变换实际上是空间域的低通滤波器)。而低频分量包含了图象的主要信息(如亮度), 而高频与之相比, 就不那么重要了, 为了达到数据压缩的目的, 在编码过程中一般都忽略高频分量。通过量化操作, 去掉高频分量。量化是产生信息损失的根源。量化操作就是将某一个值除以量化表中对应的值。由于量化表左上角的值较小, 右上角的值较大, 这样就起到了保持低频分量, 抑制高频分量的目的。从统计学角度可以发现,  $8 \times 8$  的 DCT 数据块中大部分系数为 0, 只有少数非 0 系数。因此, 在图像解码过程中, 如果完成一个  $8 \times 8$  的系数 Huffman 解码, 再进行反量化运算, 那就必然存在很多无效的乘 0 运算, 从而增加了一些不必要的乘法运算和存取操作。对其实际解码优化中, 把 Huffman 解码和反量化结合运算, 完成一个 Huffman 解码, 如果这个系数的值不等于 0, 就进行一次反量化运算; 如果该 Huffman 码系数是 0, 就存入数据缓存区, 以便进行反离散余弦变换(IDCT)运算。这样大大减少了乘法运算和存取操作。这种 Huffman 解码和反量



和反量化结合运算优化，与传统方法比较，不仅提高了运算速度，而且占用存储资源较少。

除此之外，我们对 Huffman 解码部分做了以下几方面的优化：

(1) 修改 Huff\_EXTEND()的实现方式，将查表改为计算

在原解码程序中，该函数是一个宏定义，用于取得量化后的系数：

```
#define Huff_EXTEND(x,s) ((x) < (1<<((s)-1)) ? (x) + (((-1)<<(s)) + 1) : (x))
```

其中 extend\_test[]和 extend\_offset[]分别是两个静态数据结构：

```
static const int extend_test[16] =
{ 0, 0x0001, 0x0002, 0x0004, 0x0008, 0x0010, 0x0020, 0x0040, 0x0080,
  0x0100, 0x0200, 0x0400, 0x0800, 0x1000, 0x2000, 0x4000 };

static const int extend_offset[16] =
{ 0, ((-1)<<1) + 1, ((-1)<<2) + 1, ((-1)<<3) + 1, ((-1)<<4) + 1,
  ((-1)<<5) + 1, ((-1)<<6) + 1, ((-1)<<7) + 1, ((-1)<<8) + 1,
  ((-1)<<9) + 1, ((-1)<<10) + 1, ((-1)<<11) + 1, ((-1)<<12) + 1,
  ((-1)<<13) + 1, ((-1)<<14) + 1, ((-1)<<15) + 1 };
```

我们将程序中的 s=Huff\_EXTEND(r,s)宏，用计算的方式实现：

```
if (r < (1<<((s)-1))
    s = r;
else
    s = r + (((-1)<<(s)) + 1;
```

虽然这样可能会增加一点代码量，但是和使用查表的方式，每次访问内存或者 cache 所产生的惩罚周期相比，这点代码量是微不足道的，通过测试比较，这样修改也确实大大减少了代码执行时间。

(2) 将结果缓冲区每次哈夫曼解码前清 0，减少查表次数

由于量化后  $8 \times 8$  的分块矩阵数据在内存中的存放顺序是 Zig-Zag 的，所以为

了对应原来的存储顺序，程序中建立了一个数组，专门用于对应存储位置：

```
const int jpeg_natural_order[64] = {
    0, 1,  8, 16,  9, 2,  3, 10,
    17, 24, 32, 25, 18, 11, 4,  5,
    12, 19, 26, 33, 40, 48, 41, 34,
    27, 20, 13, 6,  7, 14, 21, 28,
    35, 42, 49, 56, 57, 50, 43, 36,
    29, 22, 15, 23, 30, 37, 44, 51,
    58, 59, 52, 45, 38, 31, 39, 46,
    53, 60, 61, 54, 47, 55, 62, 63};
```

在解码程序中，每一次反量化后取得的数据，我们都要通过查找这张表来取得存贮位置，然后再根据这个存储位置将数据写入缓冲区：

```
(*block)[jpeg_natural_order[k]] = (JCOEF) s;
```

也就是说，对每一个  $8 \times 8$  分块数据，我们都要查表 64 次，即访问 64 次 Cache 或者内存。前面已经讨论过，经过量化后的数据有这样一个特点，大量的 0 系数会出现在高频部分，整个矩阵非零数据只有少数几个。鉴于此我们可以这样处理，在每次对一个分块做哈夫曼解码之前，我们可以先将缓冲区清零，然后在解码过程中只写量化后的非零数据，这样的话，我们就可以节省许多查表的工作，虽然这样会导致对非零数据位置写两次的结果，但是和节省的查表的代码比起来要小多了。

### 5.3 IDCT 算法的 DSP 快速实现

根据章节 2.2.1.3 中的推导，8 点一维 IDCT 可用如下公式流程来实现：

$$\begin{aligned}
 (1) \quad & F_{16}(0) = 5.6568 * S_8(0), F_{16}(1) = 3.9231 * S_8(1), F_{16}(2) = 3.6956 * S_8(2) \\
 & F_{16}(3) = 3.3259 * S_8(3), F_{16}(4) = 2.8284 * S_8(4), F_{16}(5) = 2.2223 * S_8(5) \\
 & F_{16}(6) = 1.5307 * S_8(6), F_{16}(7) = 0.7804 * S_8(7)
 \end{aligned}$$

$$(2) \ a_0 = \frac{1}{16}F_{16}(0), a_1 = \frac{1}{8}F_{16}(4), a_2 = \frac{1}{8}F_{16}(2) - \frac{1}{8}F_{16}(6), a_3 = \frac{1}{8}F_{16}(2) + \frac{1}{8}F_{16}(6)$$

$$a_4 = \frac{1}{8}F_{16}(5) - \frac{1}{8}F_{16}(3), a_5 = \frac{1}{8}F_{16}(1) + \frac{1}{8}F_{16}(7), a_6 = \frac{1}{8}F_{16}(3) + \frac{1}{8}F_{16}(5)$$

$$a_7 = a_5 - a_6, a_8 = \frac{1}{8}F_{16}(1) - \frac{1}{8}F_{16}(7), a_9 = a_5 + a_6$$

$$(3) \ b_0 = a_2 * h_4, b_1 = -(a_4 * h_2 + a_8 * h_6), b_2 = a_7 * h_4, b_3 = -a_4 * h_6 + a_8 * h_2$$

$$(4) \ c_0 = b_3 - a_9, c_1 = c_0 - b_2, c_2 = a_0 - a_1, c_3 = b_0 - a_3, c_4 = a_0 + a_1$$

$$(5) \ d_0 = c_2 + c_3, d_1 = c_4 + a_3, d_2 = c_2 - c_3, d_3 = c_4 - a_3, d_4 = b_1 - c_1$$

$$(6) \ f(0) = a_7 + d_1, f(1) = d_0 + c_0, f(2) = d_2 - c_1, f(3) = d_3 - d_4$$

$$f(4) = d_4 + d_3, f(5) = d_2 + c_1, f(6) = d_0 - c_0, f(7) = d_1 - a_7$$

其中:  $h_2 = 2\cos\frac{\pi}{8}, h_4 = 2\cos\frac{\pi}{4}, h_6 = 2\cos\frac{3\pi}{8}$ 。

对计算  $b_1, b_2$  的算式变形为:

$$\begin{aligned} b_1 &= -h_6 * (a_4 + a_8) - a_4 * (h_2 - h_6) \\ b_2 &= -h_6 * (a_4 + a_8) + a_8 * (h_2 + h_6) \end{aligned} \quad (5-1)$$

则可以节约一次乘法。

从上面的运算步骤可以看出,一维 IDCT 运算已经转化成了一系列简单的加减法及与常数相乘的乘法运算。简化了 IDCT 运算的复杂度,并大大降低了运算量。

需要注意的是,经过 16 次上述一维 8 点 IDCT 运算后得到的  $8 \times 8$  系数矩阵要除以 8,才是符合式(2-5)的 IDCT 系数。

下面我们讨论 IDCT 算法的 DSP 实现的优化措施。

在 4.5 节我们介绍了 EMAC 指令功能,它在 IDCT 算法的计算中同样起到了很重要的作用。我们结合 AA&N 算法原理和 DSP 的结构及指令特点,在以下几个方面进行探索。

- (1) 合并乘法和加法运算,以便利用乘法累加器。算法依托于 DSP 的加法指令及双操作数乘法累加指令,尽量不用乘法指令。所以我们要把 IDCT 快速算法中计算  $b_1, b_2$  的算式转化为式(5-1),这样乘法次数就可以减少一次。在进行乘累加运算时同时可以进行寄存器间数据传递,为后面的运算准备数据。

- (2) 在定点运算中，小数乘法的结果需要左移一位消除多余的符号位，通过设置 MACSR 为 0X20，使相乘结果自动左移一位实现可节省计算量，如果再结合 MAC 指令，则将相乘、累加和移位在一个指令周期内完成。
- (3) 合理使用 MCF5329 的累加器 ACCx，用累加器暂存运算中的乘法结果，以减少加法前的取数时间。比如在计算  $b_1$  的时候，先将乘法运算  $-h_6 * (a_4 + a_8)$  的结果暂存在累加器中，这样在做加法运算时省取了取被加数的指令时间。
- (4) 灵活使用辅助寄存器间接寻址指令，在存、取操作数的同时利用前递减和后递增寻址方式修改操作数地址。
- (5) 巧妙安排中间结果的排列顺序，使得当前的运算结果位于恰当的位置，便于下一级运算的读数操作，尽量不浪费时间浪费在数据传递和排序上。
- (6) 计算 16 次一维 8 点 IDCT 才能得到二维 IDCT 系数，在此过程中不要用循环方法，以避免耗时的判断和跳转指令。

采用不同方法计算 8 点 IDCT 的时间见表 5-1 所示。

表5-1 DSP 计算 8 点 IDCT 时间

	Loffler 算法	AA&N 算法	AA&N 优化算法
时钟周期数	807	620	321
t/μs	3.4	2.8	1.3

## 5.4 内插算法的实现

本文对数据的内插是简单的线性内插。内插后的数据是由内插前的数据与其相邻值做加权平均得到的。

对 h2v2 这种亚抽样后的数据进行内插是将水平方向和垂直方向上的采样率都提高一倍。也就是说，内插前的一个点对应于内插后的相邻的四个点。这种内插方法图示于图 5-9 中。

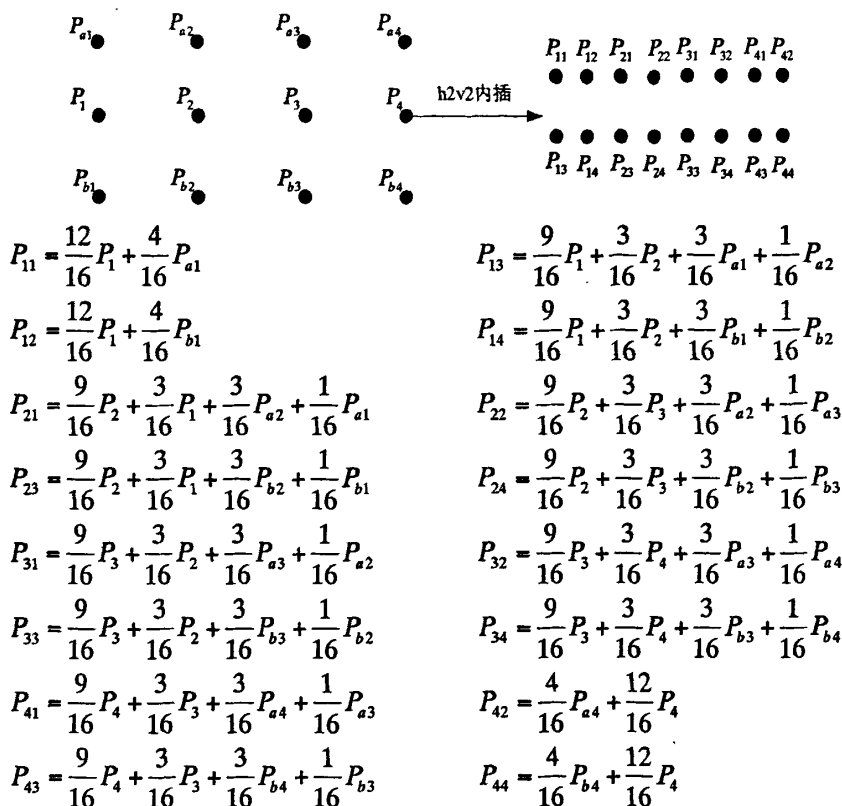


图5-9 h2v2 内插

由图 5-9 可以看出内插后的点是由内插前的点及其相邻三个点分别用 9/16, 3/16, 3/16, 1/16, 加权平均得到的。而对于第一列及最后一列的各点要单独处理。

由此可以看出 JPEG 范例中使用亚抽样与内插算法是非常粗糙的, 但是这些算法对范例中的 h2v1、h2v2 等亚抽样比的数据的处理得到的结果是很令人满意的, 而且这种简单的算法节约了很大一部分计算量。若用户使用 h4v1, h3v1 等亚抽样比率, 则需要改进亚抽样及内插算法。

## 5.5 色彩空间转换的实现与优化

在本项目中, JPEG 图片经解码后, 需要在 LCD 屏幕上显示。因 LCD 采用的是 RGB 色彩空间, 而 JPEG 文件采用 YUV 色彩空间, 所以还需要对内插后的数据进行 YUV 到 RGB 色彩空间的转换。转换公式为:

$$\begin{aligned} R &= Y + 1.402(Cr - 128) \\ G &= Y - 0.34414(Cb - 128) - 0.71414(Cr - 128) \\ B &= Y + 1.772(Cb - 128) \end{aligned} \quad (5-2)$$

对内插算法和色彩空间转换我们全部用汇编语言进行实现。

## 5.6 测试及结果分析

解压显示的时候可以将压缩数据从 flash 读到内存，边解码边将解码出来的数据传送到显示缓冲中去。图 5-10 是一幅 240×320 的 JPEG 图片经解码后在 LCD 上显示的结果。



图5-10 解码后的 JPEG 图像在 LCD 上显示

## 5.6.1 算法复杂度的测试与分析

### 5.6.1.1 时间复杂度

对一幅 240×320 的 JPEG 图片进行解码，所用时间如表 5-2 所示。

表5-2 优化前后解码时间比较

	优化前	编译器优化(O3)	手工优化
240×320 (24 位图)	0.212s	0.199s	0.120s
240×320 (灰度图)	0.090s	0.089s	0.050s

在 JPEG 解码算法中，哈夫曼解码、IDCT 变换、内插和色彩空间转换几个部分占了整体解码时间的 80% 以上，所以对 JPEG 解码软件部分的优化我们主要集中在这几个部分，表 5-3 列出了各个模块优化以后的效果。

表5-3 各模块优化后效果

优化部分	解码速度提高率
解码流程	9.6%
哈夫曼解码	3.4%
IDCT 变换	26.8%
色系变换和内插	3.1%
总计	42.9%

与编码软件优化结果类似，在解码软件优化部分中，对哈夫曼解码和色系变换以及内插部分的优化不是很理想。对于哈夫曼解码模块，其本身的并行性不高，算法中条件判断很多，难以发挥流水线并行作用；对于色系变换和内插，涉及到大量数据拷贝，使解码延迟很大，这些问题有待今后进一步研究解决。

### 5.6.1.2 空间复杂度

编译后的可执行代码所占存储空间如表 5-4 所示。

表5-4 优化前后解码器所需存储空间

	优化前	优化后
解码器大小 (KB)	99.6	125.9



## 第六章 总结与展望

本项目成功地完成了 JPEG 编解码软件在 ColdFire MCF5329 上的实现, 并对编解码器在嵌入式环境中进行了优化, 达到了产品的设计要求。

本论文主要完成了以下工作:

- (1) 深入研究并分析了 JPEG 算法标准。JPEG 顺序编码步骤包括图像预处理、亚抽样、FDCT 变换、DCT 系数量化和熵编码。相应地, 解码步骤为哈夫曼解码、反量化、IDCT 变换内插和色彩空间变换。
- (2) 设计并实现了 JPEG 编码器。改进了 JPEG 编码算法流程, 实现了一种基于 DSP 乘法累加单元的 DCT 快速算法, 对色系变换模块、亚抽样模块、DCT 系数的量化模块和哈夫曼编码模块分别进行了优化。
- (3) 设计并实现了 JPEG 解码器。改进了 JPEG 解码算法流程, 实现了基于 DSP 的快速 IDCT 算法——AA&N 算法, 对哈夫曼解码模块、量化模块、内插模块和色彩空间转换模块分别进行了优化。
- (4) 对编解码器的性能作了测试与分析, 并指出了今后工作的方向。对一幅大小为  $240 \times 320$  的 24 位真彩色图片进行编解码所需的时间分别为 0.23s 和 0.12s, 达到了设计需求。

由于经验的缺乏和时间仓促, 设计的编解码器比较简陋, 还有很多需要加以完善和改进的地方, 今后一段时间内的工作可以概括为以下几点:

- (1) 将编解码软件全部用汇编语言实现, 这样软件运行的效率将会得到极大地提高。
- (2) 对哈夫曼编码和解码进行深入研究, 找到合适的优化办法以提升其性能。
- (3) 结合硬件平台特点对软件进行优化和改进。如图像数据搬运的工作可以由 eDMA 来完成, 可以降低编解码时间。

JPEG200 是一种功能更强大、效率更卓越的静止图像压缩标准。与 JPEG 不同, JPEG2000 基于小波变换, 采用当前最新的嵌入式编码技术, 在获得优于目前 JPEG 标准压缩效果的同时, 生成的码流有较强的功能, 可应用于多个领域。

与目前流行的 JPEG 标准相比, JPEG2000 提供了更为优异的压缩性能, 更能满足人们的多方面的需求, 与此同时, JPEG2000 的算法相比 JPEG 复杂了很多, 对硬件资源的要求很高, 这也阻碍了 JPEG2000 的应用。随着高性能 DSP 处理器的涌现, 使得实时实现 JPEG2000 的编解码成为可能。另外一方面, 随着人们对 JPEG2000 算法的研究越来越多, 采用更为有效的实现方式, 也是提升软件性能, 实现实时目标的一个很好的途径。总之, JPEG2000 代表数字图像压缩的最新技术, 在未来数字图像领域必然会大有用武之地。

## 致 谢

在此，谨向所有帮助我的老师和同学表示衷心的感谢。

首先我要感谢我的导师林水生教授。林老师教给我的不仅仅是专业知识，更多是研究的方法和对科学严谨的态度，以及团队的思想 and 相互协作的能力。林老师的渊博学识，和对事业的执着追求，深深感染了我，激励我不断学习。在这里，我再次表示感谢，谢谢您对我的栽培，谢谢您对我的关心和帮助。同时，感谢李广军教授，他渊博的学识和脚踏实地的科研作风深深地感染了我，使我体会做一名研究人员需要的勤奋、执着和认真。

此外感谢郭志勇老师和阎波老师。在毕业设计过程中，他们对我的课题研究提出了许多宝贵意见，并给予了具体的指导。

感谢教研室的所有老师和同学。在与你们的学习和讨论过程中，使我学到了许多知识，非常感谢你们给我的支持、帮助和建议。

感谢我的父母和亲人们，你们的关心、支持和鼓励是我最大的学习动力和幸福的源泉，谨以此文献给你们！

最后，衷心的感谢为评阅本论文而付出辛勤劳动的各位老师。

再次向所有爱护、关心我的老师、同学、朋友们表示衷心的感谢！

## 参考文献

- [1] David.Taubman.JPEG2000——Image Compression undamentals,Standards and Practice. Kluwer Academic Publishers, 2001
- [2] 陈天华.数字图像处理. 北京: 清华大学出版社, 2007 年 6 月
- [3] 肖自美. 图像信息理论与压缩编码技术. 中山: 中山大学出版社, 2000 年 3 月
- [4] 吴乐南.静止图像压缩标准新进展.北京: 电子工业出版社, 1995 年 3 月
- [5] G.K.Wallace.the JPEG Still Picture Compression Standard.IEEE Trans.Consumer Electronics, Feb.1992, Vol.38, No 1
- [6] ISO/IEC JTC 1/SC 29/WG 1.ISO/IEC FDIS 15444-1:Information Technology——JPEG 2000 Image Coding System:Core Coding System.2000
- [7] 景晓军.图像处理技术及其应用.北京: 国防工业出版社, 2005 年 8 月
- [8] CCITT,Information Technology——Digital Compression and Coding of Continuous-Tone Still Images——Requirements and Guidelines,Recommendation T.81.1992.9
- [9] 郝杰, 吴元清, 郑榕.实用多媒体技术及其 C 语言实现.北京: 电子工业出版社, 1995 年 11 月
- [10] 薛永林, 刘珂, 李凤亭.并行处理 JPEG 算法的优化.电子学报, 2002 年第二期
- [11] S.Winograd.Some bilinear Forms whose Multiplicative Complexity depends on the Field of Constants.Math.System Theory,1977, Vol.10
- [12] M.T.Heidemann and C.S.Burris.Multiply/Add Tradeoffs in Length  $2^n$  FFT Algorithms.Proc. IEEE Int'l Conf.ASSP, ICASSP-85 Tampa, March.1985
- [13] W.A.Chen, C.Harrison, S.C.Fralick.A Fast computational Algorithm for the Discrete Cosine Transform.IEEE Tran.on Commu., Vol.COM-25, No.9, Sept.1997
- [14] Z.Wang.Fast Algorithms for the Discrete W-Transform and for the Discrete Fourier Transform.IEEE Tran.ASSP, Vol.ASSP-32, NO.4, Aug.1984
- [15] Byeong Lee.A new Algorithm to Compute the Discrete Cosine Transform.IEEE Tran.ASSP, Vol.ASSP-32, No.6, Dec.1984
- [16] M.Vetterli, H.Nussbaumer.Simple FFT and DCT Algorithms with Reduced Number of Operation.Signal Processing(North Holland), Vol.6, No.4, Aug.1984
- [17] N.Suehiro, M.Hatori.Fast Algorithms for the DFT and other Sinusoidal Transforms.IEEE Tran. ASSP-34, No.3, June.1986
- [18] H.S.Hou.A Fast Recursive Algorithm for Computing the Discrete Cosine Transform.IEEE Tran.ASSP, Vol.ASSP-35, No.10, Oct.1987
- [19] Christoph Loeffler, Adrian Ligtenberg, George S.Moschytz.Practical Fast 1-D DCT Algorithm with 11 Multiplications.Conf, Int'l, IEEE, Assp, 1989
- [20] William B. Pennebaker, Joan L. Mitchell. JPEG Still Image Data Compression Standard. Van Nostrand Reinhold, 1993.
- [21] Freescale.MCF5329 Reference Manual Rev.0.1.Feb.2006
- [22] Freescale.ColdFire Family Programmer's Reference Manual Rev.2.Jun.2001
- [23] 李品皎, 王爱侠, 张广渊.ColdFire 系列 32 位微处理器与嵌入式 Linux 应用.北京航空航天大学出版社, 2005

- 
- [24] 刘峥嵘, 张智超, 许振山等. 嵌入式 Linux 应用开发详解. 北京: 机械工业出版社, 2005
  - [25] Eric Hamilton. JPEG File Interchange Format. Version 1.02, 1992
  - [26] 蔡士杰, 岳华, 刘小燕. 连续色调静止图像的压缩与编码——JPEG. 南京: 南京大学出版社, 1995
  - [27] Thomas G. Lane. Libjpeg.doc, 1994-1998. <http://www.ijg.org>
  - [28] 罗凤武. 基于 MCF5272 的 JPEG 解码算法的设计与实现. 成都: 电子科技大学, 2006
  - [29] 齐美彬, 杨艳芳, 蒋建国. JPEG 编码算法的 DSP 优化实现. 合肥工业大学学报 25 卷 4 期, 2002 年 8 月
  - [30] 张雄伟, 陈亮, 徐光辉. DSP 芯片的原理与开发应用. 北京: 电子工业出版社, 2003
  - [31] 邢霄飞. 基于 C6000 系列 DSP 的 JPEG 图像压缩技术研究. 成都: 四川大学, 2006

## 作者简介

### 个人简历:

周鹏斌, 男, 汉族, 1981 年 3 月出生于湖北省安陆市;

2000 年 9 月~2004 年 7 月, 就读于桂林电子科技大学通信与信息工程学院通信工程专业, 获学士学位;

2005 年 9 月~2008 年 6 月, 就读于成都电子科技大学通信与信息工程学院通信与信息系统方向, 攻读硕士学位。

### 项目研究经历:

基于 MCF5329 的 Screenphone 的语音处理模块设计与实现

基于 MCF5329 的 JPEG 编解码器设计与实现

### 发表论文:

周鹏斌, 林水生, 李广军。电子科技大学电子科学技术研究院第三届学术交流会《DSP 软件开发中汇编优化的手段与技巧》, 2007 年 12 月。

# 基于JPEG标准的图像处理及其在MCF5329上的实现

作者：[周鹏斌](#)

学位授予单位：[电子科技大学](#)

本文链接：[http://d.g.wanfangdata.com.cn/Thesis\\_Y1378064.aspx](http://d.g.wanfangdata.com.cn/Thesis_Y1378064.aspx)

授权使用：同济大学图书馆(tjdxstg)，授权号：0a97ea3e-0f5b-46bf-8e04-9dc701481601

下载时间：2010年8月3日