嵌入式通用处理器的 MP3 解码软件优化方法

陈微微，付宇卓，赵　峰

上海交通大学 微电子学院 上海 200030

E-mail: {chenweiwei,zhaofeng,fuyuzhuo}@ic.sjtu.edu.cn

摘 要：本文提出了一套嵌入式软件的优化流程，包括算法优化、实现优化和平台优化三个阶段。基于此流程，具体介绍了在通用处理器 ARM 平台上实现 MP3 解码器软件优化的方法。通过软、硬件仿真验证，最终优化结果：实时解压缩率 128kbps、采样率 44.1kHz 的立体声 mp3 码流要求运算速度 26.2MIPS，存储空间 70k 字节。

关键词：嵌入式系统、软件优化、MP3 解码

Embedded Software Optimization for MP3 Decoder Implemented on General-Purpose Embedded Processor

Weiwei CHEN, Yuzhuo FU, Feng ZHAO

Shanghai Jiaotong University, School of Microelectronics, Shanghai 200030

E-mail: {chenweiwei,zhaofeng,fuyuzhuo}@ic.sjtu.edu.cn

Abstract: This paper proposes a software optimization flow on embedded platform, which mainly includes algorithm optimization, implementation optimization and platform-based optimization. This flow is applied to the optimization of the MP3 decoder on the low power general-purpose embedded processor ARM platform. The last optimized decoder requires 26.2MIPS and 70Kbytes memory space to decode 128Kbps, 44.1Hz joint stereo MP3 format file in real time.

Key words: Embedded System, Software Optimization, MP3 Decoder

1．前言

随着集成电路设计与工艺水平的突飞猛进，嵌入式设备的功能以同样惊人的速度发展，例如手机从简单的通话功能到现在的多功能为一体的智能手机，从而引出大量嵌入式软件的研发工作。Shannon’s Law 预测了算法复杂度的增长率将快于 Moore’s Law 所预测的处理器性能的增长趋势[6]。功能强大、低功耗、实时性成为衡量嵌入式系统优劣的主要指标，除了硬件电路设计的改进，软件也必须关注性能的问题。针对不同嵌入式平台的应用软件及相关实现算法的优化工作将成为嵌入式系统设计的一项重要工作。

本文将描述嵌入式应用软件的优化流程，并由此流程提出基于 ARM 平台的 MPEG-I Layer3(MP3)解码算法的软件优化方法。

2．嵌入式应用软件优化流程

由于嵌入式平台的特性限制，对于软件的大小和运行速度有比较高的要求，软件开发过程中需要着重进行优化工作。

优化工作开始之前，需要先根据系统要求，对软件的功能进行划分,统计性能并同时了解目标平台的特性。

嵌入式软件优化过程分为三个阶段：算法优化、实现优化和平台优化。算法优化关注于设计或选取有效的软件功能实现算法，例如实现数组排序是使用冒泡法还是快速排序法，这阶段的工作对目标平台透明，只需考虑嵌入式平台最基本的特性—内存空间有限、时钟主频较低，可选用各种高级语言，如 C、Java 语言实现，这阶段完成后的结果可以移植到各种嵌入式平台；实现优化针对编译器特性和目标平台特性，在代码书写的过程中选取较优的方式和风格提高软件性能，这阶段对平台半透明，代码书写的风格会考虑嵌入式平台指令的特点；平台优化完全基于汇编语言，针对平台量身定制软件使之与平台得到最好的匹配。

优化完成后通过在硬件平台上的运行来验证软件的正确性并测试其性能指标。

基金项目：国家 863 个人信息处理终端 SoC（2003AA1Z1350）资助。作者简介：**陈微微**，1982 年 7 月生，女，硕士研究生，研究方向为嵌入系统、微处理器系统设计；**付宇卓**，1968 年生，男，博士，教授，研究方向为芯片系统结构设计、高性能并行计算、多媒体同步模型研究；**赵峰**，1976 年生，男，博士，讲师，研究方向为微处理器系统设计。

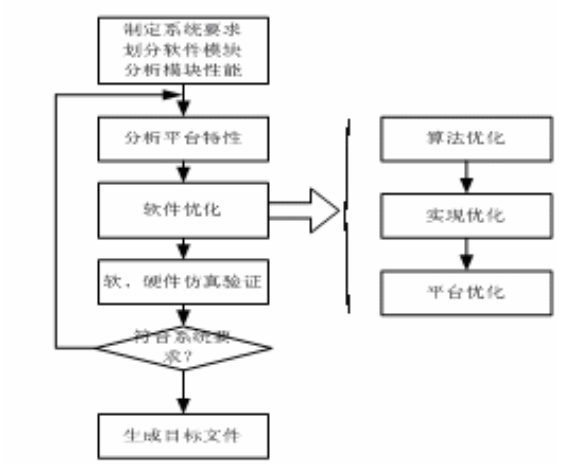

图 2.1 嵌入式软件开发优化流程

Fig 2.1 Optimization Flow for Embedded Software

3．MP3 解码算法软件优化

3. 1 MP3 解码算法简介

MPEG-I Layer3(MP3)编码标准，是国际标准化组织(ISO)运动专家组(MPEG)音频/视频编码标准之一，它是一种以不同的压缩比率、较小音质损失的代价对音频文件进行压缩的格式，压缩率为 10:1 左右[4]。

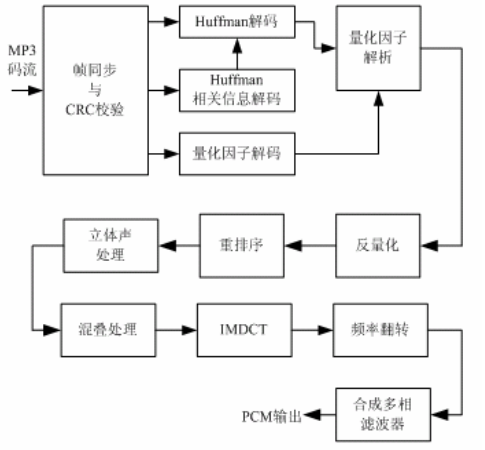

图 3.1MP3 解码流程

Fig 3. MP3 Decoding Process

MP3 编码的主要方法是在频域上对音频文件内容进行编码压缩，而解码过程是还原频域的内容再变换成原始的时域音频信号。按照 ISO/IEC11172-3 标准，MP3 解码算法分为同步与校验、Huffman 解码、比例因子解析、反量化、重排序、立体声处理、混叠处理、改进的反离散余弦变换(IMDCT)、频率翻转及合成多项滤波器等十个部分。

优化工作开始前，先分析每个部分的运算代价。表 3.1 表明了本文所采用的未经优化的 MP3 解码软件[5]在 ARM 开发套件 (ADS) 软件仿真平台上测试的结果。可以看到，Huffman 解码和反量化、IMDCT 与频率翻转、合成多项滤波器三部分占用了最多的运算时间，这三部分是优化工作的重点。

mp3 文件 压缩率	64kbps	128kbps	192kbps	320kbps
帧同步与 CRC	0.47%	0.44%	0.35%	0.35%
量化因子解析	0.01%	0.37%	0.30%	0.01%
Huffman 解码+反量化	***10.53%***	***11.17%***	***17.85%***	***24.38%***
立体声处理	0.00%	4.56%	3.62%	0.00%
重排序	0.00%	0.00%	0.00%	0.00%
混叠处理	5.44%	5.55%	4.41%	4.47%
IMDCT+频率翻转	***27.95%***	***21.23%***	***28.42%***	***25.15%***
合成多项滤波	***55.59%***	***56.68%***	***45.05%***	***45.64%***
总计	100%	100%	100%	100%

表 3.1 各种压缩比率 mp3 文件解码各部分时间复杂度比例表

Table 3.1 Profiling for Each Part of the MP3 Decoder Algorithm based on different encode rate.

测试表明未经优化的 MP3 软件需要 93MIPS，92.19k 字节存储空间（49.69k 字节代码空间和 42.5k 字节表空间）来实时解压缩 128kbps 44.1kHz 的 mp3 码流。

3. 2 算法优化

优化工作的第一步是设计或选取有效的实现算法，兼顾时间复杂度和空间复杂度两个方面。

3.2.1 定长查找冗余表 Huffman 解码算法

Huffman 编码变长的特性决定常规解码是逐一读取码流与 Huffman 编码树结点进行匹配来完成。逐一比特取码流会带来大量的位运算和从内存中取数的开销。定长查找冗余表法扩充 Huffman 查找表，每次选取定长 N 比特码流(>4)作为查找索引。查找表内包含跳转指针和叶结点两类结点，当结点索引为某 Huffman 编码的前缀时结点为指针项；当索引值含某 Huffman 编码时结点为叶结点，叶结点包含解码的结果信息。由于 Huffman 编码变长，而每次取定长比特码流，故需添加那些索引值包含 Huffman 编码的结点，结点内容为所包含 Huffman 编码的解码结果。为了减小查找表的空间，利用指针结点指向下次取定长比特所扩展成的子表的起始地址，从内存组织上来看查找表仍呈线性状态。查找表利用 union 数据结构来实现，每次解码成功后需要回退相应比特

到下一编码的起始位置。假设一 Huffman 编码长度为 *l*，采用传统算法需要 *l* 次移位操作和 *l* 次比较，使用定长查找法只需$\lceil l/N \rceil$次查找和$\lceil l/N \rceil$次比较操作。

定长查找冗余表 Huffman 解码算法举例：

Huffman 编码定义

Huffman 编码	内容
1	A
001	B
010	C
011	D
0001	E

建立查找表 H

	索引值（二进制）	表项类型	表项内容	回退
0	0000	指针	3	-
1	0001	指针	7	-
2	0010	值	A	1
3	0011	值	A	1
4	0100	值	E	0
5	0101	值	B	1
6	0110	值	B	1
7	0111	值	C	1
8	1000	值	C	1
9	1001	值	D	1
10	1010	值	D	1

图 3.2 Huffman 解码举例

Fig 3.2 Example for Huffman Decoding

Huffman 编码 0101011，对应内容为“CAD”(010→C、1→A、011→D)。

每次取定长 2 比特，解码过程为：

步骤 1: 取 2 比特 ***01***01011, $(01)_2$=1, H[1]为指针，跳转到表项***7***;

步骤 2: 取下 2 比特 01***01***011, ***7+***$(01)_2$=8, H[8]为值‘C’，回退 1；

步骤 3: 取下 2 比特回退 1 比特 010***10***11, $(10)_2$=2, H[2]为值‘A’，回退 1；

步骤 4:取下 2 比特回退 1 比特 0101***01***1, $(01)_2$=1, H[1]为指针，跳转到表项***7***;

步骤 5:取下 2 比特 010101***1***, 比特数不足补 0，***7+***$(10)_2$=9, H[9]为值‘D’，回退 1；

解码结果为“CAD”。

从空间复杂度上看，MP3 解码算法中，Huffman 编码 *hcode* 最长为 *17* 比特，编码长度 *hlen* 为 *5* 比特，编码内容 *x*、*y* 各为 *4* 比特，原始 Huffman 查找表每项需 *30* 比特字节对齐(align)为 *4* 字节；由于码流直接作为地址索引，表内匹配 Huffman 编码的 *hcode* 部分可以删除，5 比特 hlen, x、y 各 4 比特、1 比特表示结点类型, 冗余表每项只需 *14* 比特字节对齐(align)为 2 字节。冗余查找表虽然表长较长，但每个表项所占空间较小，重新建表后表空间基本不变。从时间复杂度上分析，根据 Huffman 熵编码的特性，出现概率高的编码长度较短，而较短长度的编码使用定长查找冗余表法基本只需 1~2 次比较即可完成解码，最长的编码比较次数也不会超过所跳转的子表个数，当编码表较长时查找次数可看作 O(1)。

3.2.2 精简反量化系数表

Huffman 解码后得到的是量化后的频线，反量化的工作是根据量化因子的解码结果重建原始的频率线作为下一步 IMDCT 的输入。反量化公式为

$$xr[i] = sign(huffi) * abs(huffi)^{4/3} * 2^{\frac{(global\_gain-210-8sbg[i])}{4} - scalefac\_multiplier*(sf[i]+preflag*pr[i])}$$

上述公式中 2 的幂运算可把幂指数分为整数和小数两部分，2 的整数幂直接通过移位运算得到，小数幂部分只有-4/3，-1/2，-1/4，1/4，1/2，3/4 六个值，直接建表查找。反量化的运算工作主要集中在$abs(huffi)^{4/3}$的运算上。huffi 是 Huffman 解码的结果，共 8207 个值，直接对所有值的 4/3 次幂建表，表的大小将超过 32K 字节，对于嵌入式系统而言太大。经测试发现，huffi 的值一般不超过 100，几乎不超过 1000。精简反量化 huffi 的 4/3 次幂的系数表，对于频繁使用的小值区不间断编表，对于不常使用的大值区离散编表，未编入的值通过差分计算得到。考虑到精度问题，分别采用 1 阶差分和 2 阶差分方式进行实验。

一阶差分计算公式：

$Value[i] = (Value[i_u] - Value[i_d]) / step * (i - i_d) + Value[i_d]$ 其中 $i_d = i - i\%step + step$， $i_d = i - i\%step$；

二阶差分计算公式：

$Value[i] = ((Value[i_u] - Value[i_d]) - step * (step - 1) * d) / step * (i - i_d) + (i - i_d - 1) / 2 * d + Value[i_d]$ 其中 $i_d = i - i\%step + step$， $i_d = i - i\%step$，***d*** 是 ***value*** 数列的 2 阶差分值，可查表得到。

表 3.2 表明了不同离散幅度、不同离散区间分别做 1 阶差分和 2 阶差分的表大小。实验表明，采用直接编表区 1-255，离散编表区 255-8207，离散幅度 8，表的大小下降到 4.88k，反量化计算误差不超过 5.21956E-05，解码输出结果每个采样点误差不超过 1 比特。

直接编表区	离散编表区	离散幅度	1 阶差分表大小	2 阶差分表大小
1－256	257－8207	4	8.77k	16.52k
1－256	257－8207	8	4.88k	8.76k
1－512	513－8207	4	9.51k	17.02k
1－512	513－8207	8	5.75k	9.51k
1－1024	1025－8207	4	11.01k	18.02k
1－1024	1025－8207	8	7.50k	11.01k

表 3.2 反量化差分查找表统计

Table 3.2 Statistics for the Requanization Lookup Table

3.2.3 快速离散反余弦变换(IMDCT)

IMDCT 基于 32 个频率子带，每个子带含 1 个长窗块或者 3 个连续的短窗块，长窗由 18 条频线组成，产生 36 个输出；短窗由 6 条频线组成，产生 12 个输出，3 个短窗共 36 个输出。IMDCT 公式为

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos(\frac{\pi}{2n}(2i+1+\frac{\pi}{2})(2k+1))$$
$$0 \le i \le n-1$$

根据[3]提出的算法，利用两个 cos 值绝对值相同，使用积化和差的计算方法合并同类项，将乘法化为加法。 先将 N 点 IMDCT 转成 N/2 点的 DCT_IV，接着化成 N/2 个点的 sdct_II,最终转化成两个 N/4 点的 sdct_II。算法复杂度为 3N/4 次乘法，7N/4-2 次加法再加 2 个 N/4 点的 sdct_II。

3. 3 实现优化

高级语言编写程序有益于提高可读性和移植性，然而由于不同嵌入式平台具有各自的特点，在实现优化同时也要考虑目标平台指令集和构架的特性。

3.3.1 子带合成优化算法

子带合成滤波器在 MP3 解码算法中占 45%以上的运算量，有很大的优化价值。合成滤波器分为矩阵运算和 PCM 输出窗滤波两个部分，其中矩阵运算的公式为

$$V(i) = \sum_{k=0}^{31} N_{ik} S_k$$
$$N_{ik} = \cos((16+i)(2k+i)\pi/64)$$
$$i = 0 \sim 63, k = 0 \sim 31$$

本文采用[1]中所提供的算法, 这个算法中主要运算是乘加，优化的目标就是降低乘法的运算时间。我们通过定点化和双字乘加转为单字乘加操作的方法进行优化。

定点化将浮点操作变为定点操作。定点化之后，每个数使用 32 比特（4 字节）来表示，第 31 位为符号位，第 30~28 位为整数位，其余 28 位为小数位。两个 32 比特的数相乘得到一个 64 比特的结果。

在ARM指令集中存在两种指令伪码：32 比特×32 比特 → 64 比特的乘法和乘加指令smull、smlal和 32 比特×32 比特→32 比特的乘法和乘加指令mul、mla。前者指令周期为 3+m，后者为 2+m’，其中m、m’ = 0,1, …, 4, 不同的乘数和被乘数对应不同的m、m’值。虽然m和m’的值不完全一致，但从总体上而言mul、mla的指令周期比smul、smla的短。基于这个事实，我们尽量使用 32 比特相乘得到 32 比特结果的做法。分析子带合成乘法的 64 比特结果在 -7.999999996~7.999999996 之间，有效数位为第 28~59 位，第 60~63 皆为 0，第 0~27 位忽略不计。我们重新修改窗口系数表，将其舍入并移 14 位，由于窗口系数数组在定点化后低 12 位均为 0，精度损失为最后 2 位。然后在DCT计算时移动 12 位。最后在保存结果是再移动 2 位。这样总共丢失低 14+12+2=28 比特数据，两数相乘的最大误差为 2^{-14}。由于计算从 64 位的乘法变成 32 位的乘法，大大降低了指令周期数。

3.3.2 代码整体优化

该阶段优化在代码编写上考虑目标平台指令的特性。

a）减计数循环。在 IMDCT 和合成多项滤波这两个主要运算部分都含有大量的循环体及其嵌套。在实现时使用减计数结构的循环，执行速度比增计数结构要快。对于一个 N 次循环结构，将 for(int i = 0; i < n; i ++)写成 for(int i = n; i != 0; i--)的形式。表 3.3 显示了固定次数循环使用这两种不同写法的汇编结果。

for(i = 0; i < 32; i ++) *// loop body;*	*for(i = 32; i !=0; i ++)* *// loop body;*
mov r1,#0 *loop* *;loop body* *add r1,r1,#1* *cmp r1,#0x20* *bcc loop*	*mov r1,#0x20* *loop* *;loop body* *sub r1,r1,#1* *bne loop*

表 3.3 增、减计数循环结构比较

Table 3.3 Comparison for Increasing and Descending Loop

可以看到，增计数循环使用了 3 条指令：add 增加 i 值、cmp 检查 i 是否小于 64、bcc 条件分支；减计数循环只使用了 2 条指令：sub 减少 i 值和 bne 条件跳转指令。这样每次循环减计数比增计数少一条指令，同样对于不定次数循环减计数实现每次循环也可减少 1-2 个指令周期。当循环次数很多时，减计数循环的速度是最快的。

b）局部变量使用全局查找表

在函数中，将全局查找表的入口地址赋为局部变量的初值，通过该局部变量来访问查找表。而直接使用全局查找表需先找到基址，然后按偏移计算地址。所以使用局部变量可减少每次使用都取基址的操作。

3. 4 平台优化

3.4.1 动态分配空间的复用

在 MP3 解码算法中需要使用动态分配空间来保存中间计算结果和 PCM 输出值。中间计算结果包括 Huffman 解码结果、反量化结果、IMDCT 变换结果和合成多项滤波结果。其中 PCM 输出值与 Huffman 解码后的计算结果相互独立可

共享一个存储空间，使用同一块存储区域，可减少 8k 字节动态分配空间。所有相互独立的计算结果均可复用数据空间。

3.4.2 内联函数展开和内嵌汇编

MP3 解码算法中定点化乘法运算通过函数调用实现。乘法运算包括 2 个定点化的 32 位数相乘得到 64 位结果，然后移位存入另一个 32 位寄存器中。函数实现时，每实现一次移位乘法开销是 25~30 个时钟周期，但其中超过 15 个周期用于函数调用时 pc 指针以及寄存器压栈保存上。采用内联函数方式(用__inline 标记声明)，在编译时代码段被直接展开，避免了函数调用时调整 pc 指针的开销。另外，由于 arm 编译器的支持，在嵌入汇编时直接使用 C 语言变量作为汇编的操作数（x、y、low、high），在编译时可灵活调度寄存器，避免了直接使用寄存器作为汇编操作数而必须事先压栈保护该寄存器值的操作。使用内联函数结合嵌入汇编实现移位乘法，效率可提高 100%以上。下面是实现移位乘法的代码，使用内联函数后平均时钟周期为 8~10。

```
__inline int   _f_mul(int x, int y)
{
  signed int high,low;
  // low as result
  __asm{
    smull   low, high, x, y;
    movs   low, low, lsr #0x1c ;
    adc     low, low, high, lsl #0x4;
    }
    return low;
}
```

4．优化结果与结论

4.1 实验测试平台与实验结果

实验使用业界广泛使用的 ARM 开发套件(ARM Developer Suite v1.2， ADSv1.2)作为软件开发与测试平台，MP3 解码算法采用 C 语言与内嵌汇编语言结合的方式实现，实验每一步评估结果基于 ADS 的软件仿真分析。每一步优化的结果以及不同压缩率 mp3 文件解码效率见表 4.1。

本文所讨论的 MP3 解码软件不仅通过 ADS 的仿真，同时也在硬件平台上真实地运行通过并播放出旋律。

实验硬件平台为基于 ARM920T 的系统芯片，ARM920T 芯片主频为 48MHz，含有 8K 字节指令高速缓存（I-Cache）和 8K 字节数据高速缓存（D-Cache），系统芯片中除了 ARM920T 处理器外还包含 DAC(Digital-to-Analog Converter), DMA (Direct Memory Access), SPI (Serial Peripheral Interface)以及 96K 字节的 SRAM 等部件用于音频信号输出。系统芯片不包含其它硬件加速器。本文着重于论述软件的优化方法，故未采用任何相关硬件加速器。

4.2 实验结论与讨论

我们选取 ARM920T 为平台而不是运算功能强大的数字信号处理器(DSP)为平台是考虑到 ARM 平台的通用性和低功耗设计。我们将实验结果分别与[2]、[7]中基于 ARM7TDMI 和 TMS320LC54X 数字信号处理器硬件平台的 MP3 解码算法实现结果进行比较（表 4.2），可以看出本文所提出的优化流程与方法在提高软件性能方面有很好的作用。但由于本文所编写的程序主要采用 C 语言实现，在代码大小的控制上不如直接使用汇编语言。

	结果
优化前	93MIPS, 91.19k 字节内存空间 （49.69k 字节代码空间、42.5k 字节表空间） （未考虑动态分配空间）
算法优化	减少 23.4MIPS (25.2%) 减少 24k 字节表空间
实现优化	减少 18.7MIPS (20.2%)
平台优化	减少 24.7MIPS （26.6%） 减少 8k 字节动态分配空间
优化结果	26.2MIPS，70k 内存空间 (42k 字节代码空间、28k 字节表空间) （包括所有动态分配空间）

mp3 文件 压缩率/采样率	实时解码 运算效率
56kbps/22.5kHz	7.9MIPS
64kbps/48kHz	14.6MIPS
128kbps/44.1kHz	26.2MIPS
192kbps/44.1kHz	32.9MIPS
320kbps/44.1kHz	33.2MIPS

表 4.1 MP3 解码优化结果

Table 4.1 The Optimization Result

mp3 文件 压缩率/采样率	解码运算效率 与 代码空间	
	ARM920T	ARM7TDMI
128kbps/44.1kHz	26.2MIPS 70K 字节	48MIPS 49K 字节
mp3 文件	解码运算效率 与 代码空间	

压缩率/采样率	ARM920T	TMS320C549
96kbps/44.1kHz	21.7MIPS 70K 字节	52.3MIPS 23.25k 字

表 4.2 不同硬件平台 MP3 解码性能比较

Table 4.2 Comparison MP3 Decoder Performances among Different Platforms

经过算法优化、实现优化和平台优化 MP3 解码软件只需要 26.2MIPS，70k 字节内存空间（42k 字节代码空间和 28k 字节表空间）来实时解压缩率 128kbps、采样率 44.1kHz 的 mp3 码流，程序的时间和空间性能都得到了显著的提高。解码结果，每个采样点误差不超过 1 比特，信噪比大于 90db。

References：

[1] Hung-Chih Lai, "*Real-time Implementation of MPEG-1 Layer 3 Audio Decoder on a DSP Chip*", M.S. thesis. National Chiao-Tung University, Hsinchu, Taiwan.

[2] Yingbiao Yao,Qingdong Yao, Peng Liu and Zhibin Xiao, "*Embedded software optimization for MP3 decoder implemented on RISC core*"，Consumer Electronics, IEEE Transactions on Volume 50, Issue 4, Nov. 2004, pp.1244 – 1249.

[3] Szu-Wei Lee, "*Improved algorithm for efficient computation of the forward and backward MDCT in MPEG audio coder*", Circuits and Systems II: Analog and Digital Signal Processing, IEEE Transactions on,Volume 48, Oct. 2001 pp.990 – 994

[4] ISO/IEC JTC1/SC29/WG11 MPEG, IS11172-3 "*Information Technology – Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5Mbit/s, Part3: Audio*"1992.

[5] *libmad-MPEG audio decoder library*, copyright (c) 2000-2004 Underbit Technologies, Inc. www.underbit.com

[6] Harrison Lee, "*Moore's Law Meets Shannon's Law: The evolution of the Communication's Industry*", Computer Design, 2001. ICCD 2001. Proceedings. 2001 International Conference on 23-26 Sept. 2001， pp:5 – 5

[7] Haihua WU, "DSP Implementation for Real-time MPEG Audio Layer II Decoder System", Master Thesis, Shanghai Jiao Tong University, Dec 1999.

附中文参考文献：

[1] 吴海华, "MPEG Audio Layer III 实时编解码系统的 DSP 实现"，上海交通大学 硕士学位论文，分类号 TN912.3，授予时间 1999 年 12 月 1