



## MP3技术简介

- **MP3**标准采用正交镜像滤波器组把音频信号划分为**32**个等宽子带，使用**MDCT**、声道合成技术、自适应比特分配技术、自适应窗切换技术、熵编码及变速率编码的比特缓冲技术。
- **MP3**标准加大了计算复杂度，但获得更低的码率和更好的重建音质。



## MP3的数据流格式

- **PCM**音频信号进行**MP3**压缩时，以**1152**个**PCM**采样值为单位，封装成具有固定长度的**MP3**数据帧。
- 这**1152**个取样值被分成**2**个粒度组，每个粒度组包含**576**个采样值。
- 一个**MP3**数据帧分为**5**个部分：帧头、**CRC**校验值、边信息、主数据、附加数据。



## 帧头和CRC校验部分

- 帧头区分每一帧数据的开始。
- 帧头中定义了**MP3**数据的采样率、比特率，单通道/双通道信息，采用的压缩算法的参数以及其他附加信息等等。
- **MP3**的帧头有**32**比特长，包括**12**比特同步字（帧同步字“**111111111111**”）及**20**比特的帧数据信息。
- 在解码源程序中定义了结构**frame\_params**来存贮这些信息。



## 边信息(side\_ information)

- 在每帧**MP3**数据中，边信息将提供解码需要的**Huffman**表，伸缩因子信息，比特压缩时的主数据起点等。
- 在解码源程序中，定义了结构**III\_side\_info\_t**来存贮这些信息。
- 在**MP3**中，主数据(**main data**)并不一定跟随在边信息的后面。所以使用**Main data begin**这个**9**比特宽的指针，用来标志当前帧主数据的位置。



## 主数据(main data)

- **MP3**数据帧的主数据部分包括缩放因子(scalefactors), **huffman**编码数据、附加数据**3**部分。
- 其中**huffman**编码数据是**MP3**数据的主要部分。



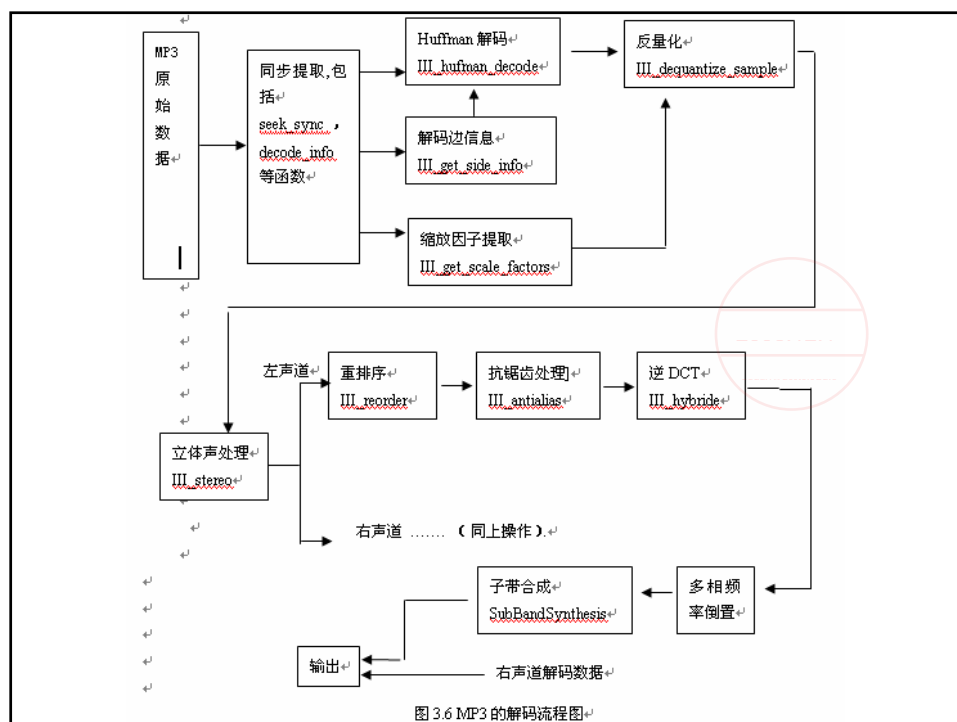
## MP3解码算法描述

- 首先检测数据流中的同步字以得到正确的帧信号。
- 再提取帧头信息，从而得到相应的解码参数，同时分离边信息和主数据。
- 边信息数据通过解码可得到霍夫曼解码信息和反量化信息，主数据就可以根据霍夫曼解码信息解码出量化前数据。



## MP3解码算法描述

- 结合帧头给出的立体声信息，对反量化结果进行立体声处理。
- 再经过变换域的计算，通过混叠处理、**IMDCT**、合成滤波处理就可以得到原始音频信号。
- 下面是MP3解码算法的框图：



## MP3解码算法描述

- 数据流同步及帧头、边信息的读取
  - ◆ 在进行解码时，也是以帧为单位，在数据流中搜索同步字，若搜索到一帧便开始解码。
  - ◆ 提取帧头信息，然后是边信息及主数据。
  - ◆ 帧头信息中包含采样率、比特率、填充位等主要信息。比特率和填充位信息用来确定每帧的帧长。

## MP3解码算法描述


- 主数据的提取
  - ◆ MP3标准中用到了数据池(bit reservoir)技术，所以当前帧的主数据不一定都在当前帧中。
  - ◆ 在解码过程中，必须结合main\_data\_end的值（帧头信息提供）来确定主数据的位置。
  - ◆ 因此，解码器中需要开辟一个缓冲区来作为数据池的存储空间，缓冲区最长为 $2^9-1=511$ 字节。

## MP3解码算法描述

- 反量化

- ◆ 反量化的目的是重建编码时经过MDCT变换输出的频域样本值。反量化基于前面步骤中所得到的huffman解码数据is[i]、缩放因子信息及边信息。

- ◆ 下面是反量化公式（反量化后的值为xr[i]）：


$$xr[i] = \text{sign}(is[i]) \times \text{abs}(is[i])^{\frac{4}{3}} \times \frac{2^{\frac{1}{4}(\text{global\_gain} - 210 - 8\text{sb}g[i])}}{2^{\text{scalefac\_multiplier} \times (\text{sf}[i] + \text{preflag} \times \text{pt}[i])}}$$

## MP3解码算法描述

- 立体声处理

- ◆ MP3中除了支持简单的单声道和立体声(相互独立声道)外，还支持更为复杂的联合立体声模式：**MS立体声(MS - stereo)**和**强度立体声(intensity stereo)**。通过帧头中的模式(mode)和模式扩展位(mode extension)来确定。

## MP3解码算法描述

- 重排序

- ◆ 反量化过程中得出的频谱值并不是按相同顺序排列的。
- ◆ 在编码的MDCT过程中，对于长窗产生的频谱值先按子带然后按频率排列；
- ◆ 对于短窗，产生的频谱值是按子带、窗、频率的顺序排列的。



## MP3解码算法描述

- 混叠重建

- ◆ 在编码的MDCT过程中为了得到更好的频域特性对每个子带进行了去混叠处理。为了得到正确的音频信号，在解码时必须进行子带的混叠重建。
- ◆ 每个子带的混叠重建由8个碟形运算组成，
- ◆ 其中cai和csi的值由MP3标准给出。





## MP3解码算法描述

- 逆向离散余弦变换(IMDCT)

- ◆ 经过混迭消除后的信号便要进行IMDCT变换。

- ◆ IMDCT的变换公式为:

$$x_i = \sum_{k=0}^{\frac{n}{2}-1} X_k \cos\left(\frac{\pi}{2n}\left(2i+1+\frac{n}{2}\right)(2k+1)\right) \quad , \text{for } i=0 \sim n-1$$



## MP3解码算法描述

- 频率反转和子带合成

- ◆ 频率的反转是对IMDCT的子带输出值进行反相处理，用来补偿编码时为提高MDCT变换效率而进行的频率反转。

- ◆ 子带合成是IMDCT变换后的一个通道中32个子带的样值，经过一系列运算还原出32个PCM数字音频信号的过程。



## MP3标准解码代码在CCS下的实现

- 在PC上实现MP3
- 将标准的C代码移植到CCS下运行
- 压缩MP3解码程序对存贮器的需求
- 浮点版本的MP3解码程序（标准版）的运算量统计



## 在PC上实现MP3

- MP3的标准C解码参考源程序可以从网上下载，这是PC上的浮点数运算版本。
- 首先在VC6下完成MP3解码软件的编译、连接，并在PC上完成MP3的解码操作。
- 通过命令行输入需要解码的MP3文件和解码输出的PCM文件。
- 该输出文件是一个16bit的线性音频数据文件，但没有文件头，播放器无法直接打开。



## 将标准的C代码移植到CCS 下运行

- 在CCS的仿真环境下建立工程文件，并编译、连接、运行，直到能使用DSP完成解码运算。（...\\probe\_C\_mp3）
- 这时我们不考虑运行的时间，而只关心能否正常完成解码运算。
- 为了进一步接近实际用法，可以将原来的数据文件输入/输出改为探针方式。



2024-12-13  
qguan@uestc.edu.cn

## 将标准的C代码移植到CCS 下运行

- 对霍夫曼码表数据文件（huffdec.txt）的处理
  - ◆ 在huffman.c 文件文件中，我们用一个静态数组来替代该码表数据的文件操作，并将该定义的数组放到一个头文件中（huff\_tab.h）。
  - ◆ 然后将原来定义的结构数组变量ht[HTN]改为结构指针\*ht。再改写读表函数



read\_decoder\_table():

## 将标准的C代码移植到CCS 下运行

- 修改decode.c中的initialize\_huffman () 函数，去除霍夫曼码表文件操作。
- 将合成滤波器的系数改为静态数组，并放到头文件de\_windows.h。修改有关该滤波器系数的文件操作。
- 在de\_decode.c文件中的out\_fifo ()函数是将解码后的结果保存到文件中。我们将使用探针工具来保存解码后的数据。



## 将标准的C代码移植到CCS 下运行

- 对输入数据文件的处理，也是采用探针工具实现数据的输入。
- 建立一个输入缓冲File\_bs\_buf，每次MP3原始数据将被读到这个缓冲。
- 在CCS的simulator下仿真运行，得到探针输出文件（stereo\_sim\_float.ccs\_probe\_c.ccs），然后将其转换为二进制数据文件，并加上WAV文件头，就可以使用媒体播放器试听。



2006.12.13

qguan@uestc.edu.cn

## 标准MP3解码程序对存贮器的需求

- 虽然在上面的步骤中能够在CCS的SIMULATOR下正确解调，但通过MAP文件，可以发现，该工程文件需要的存贮器相当大。
- 程序空间大约需要26K字（0x659a），数据空间约需要51K字（0xcbf0）。

2006.12.13

qguan@uestc.edu.cn

## 压缩标准MP3解码程序对存贮器的需求

- 减少输入数据缓存
  - ◆ 解码程序为MP3原始输入数据设置了一个输入缓存，每当需要解码操作时，先从该缓存读取数据，若缓存中没有数据则先从输入文件读数据到缓存。
  - ◆ 数据代码中使用的输入数据缓存设置为4K字，两个这种缓存。可以将其减小到1K字。

## 压缩标准MP3解码程序对 存贮器的需求

- 删除初始化代码，使用静态变量，如在 `decode.c` 中的 `III_antialias()` 函数。
- 通过统计堆栈和系统堆的使用，我们为系统保留了 `0x2200` 字的系统堆共代码申请内存操作，保留 `0x800` 字的空间为堆栈段。
- 将动态内存申请的操作去掉，改为静态数组，以便统计存贮器消耗量。\*



## 压缩标准MP3解码程序对 存贮器的需求

- 采用内存复用技术进一步压缩存贮器。
- 近一步分析代码，我们发现有些中间结果使用的存贮器是可以复用的。
- 下面的表格为主要数组的定义、大小和功能，其中 `SBLIMIT=32`，`SSLIMIT=18`，`ch` 为 0 或 1 表示左/右声道



程序中定义的数组	功能说明
<code>static long int is[SBLIMIT][SSLIMIT]</code>	哈夫曼解码后的结果放在is中
<code>static double lr[2][SBLIMIT][SSLIMIT], ro[2][SBLIMIT][SSLIMIT]</code>	反量化的结果在ro[ch]中（输入数据在is中）。立体声处理时，输入数据在ro中，输出数据在lr中
<code>static double re[SBLIMIT][SSLIMIT]</code>	混序处理函数使用，输入数据lr[ch]，输出re
<code>static double hybridIn[SBLIMIT][SSLIMIT]</code>	抗锯齿处理使用，输入数据re，输出数据在hybridIn
<code>static double hybridOut[SBLIMIT][SSLIMIT]</code>	IMDCT变换处理函数使用，输入数据hybridIn，输出数据在hybridOut中
<code>static double polyPhaseIn[SBLIMIT]</code>	子带合成处理时，数据先从hybridOut到polyPhaseIn中，然后进行子带合成滤波，完成数据解码。
<code>int is_pos[576]</code> 和 <code>double is_ratio[576]</code>	只在立体声解码函数III_stereo中使用

## 压缩标准MP3解码程序对 存贮器的需求

- 根据上页的统计，我们将这些数组改为指针操作，让可以被覆盖的存贮器重复使用。
- 主要数组的复用情况安排如下面所示。
- 这样，其最大内容消耗为：

$2304+2304+1152+576=6336$  bytes



占有字节数	数组使用情况		
$(2 \times 32 \times 18) \times 2 = 2304 \text{ bytes}$	lr 数组		
$(2 \times 32 \times 18) \times 2 = 2304 \text{ bytes}$	ro 数组	re数组 ( $32 \times 18 \times 2$ )	
		hybridIn数组 ( $32 \times 18 \times 2$ )	
$(32 \times 18) \times 2 = 1152 \text{ bytes}$	is 数组	hybridOut数组 ( $32 \times 18 \times 2$ )	is_ratio数组 ( $576 \times 2$ )
		polyPhaseIn 数组 ( $32 \times 2$ )	is_pos数组 (576)

## 压缩标准MP3解码程序对 存贮器的需求

- 通过MAP文件可以看出，这个代码包括程序和数据空间小于64K字。
- 其中程序空间约17K字(0x458d)，数据空间约35K字(0x8b68)。
- 数据空间较大的原因是我们将huffman表，合成滤波器的系数以及一些数据常量都放了进去。

(...\de\_mp3\_q15\_sim)





## 浮点版本的MP3解码程序的运算量统计

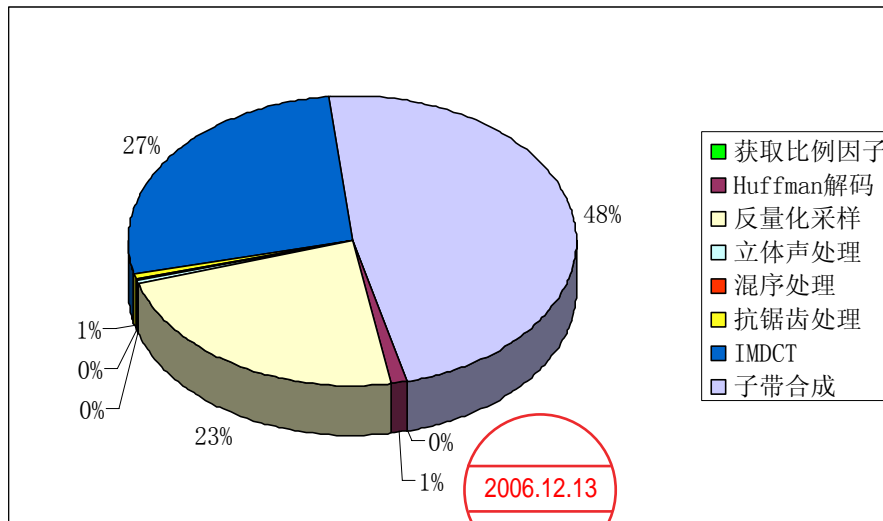


- 根据MP3的标准，每帧数据包含1152采样点，原始音频数据采样率为44.1KHz，则每帧数据的解码必须在26.12ms内完成。
- 若采用100MHz的'C5402，每帧数据解码最多可以有2.6M个机器周期。
- 下面给出了使用定点DSP仿真浮点运算时，一帧数据解码的主要函数的消耗的机器周期数。



函数名	单次调用机器周期个数	每帧数据调用次数（立体声）	功能
III_get_scale_factors	3115	4	获取比例因子
III_huffman_decode	236686	4	Huffman解码
III_dequantize_sample	5204569	4	反量化采样
III_stereo	83243	2	立体声处理
III_reorder	31457	4	混序处理
III_antialias	120729	4	抗锯齿处理
III_hybrid	189468	128	IMDCT
SubBandSynthesis	595742	72	子带合成
总的机器周期数			89698038

2006.12.13  
qguan@uestc.edu.cn



## 浮点版本的MP3解码程序的运算量统计

- 分析这些函数的机器周期数，我们可以发现，反量化采样函数III\_dequantize\_sample、反DCT变换函数III\_hybrid以及子带合成函数SubBandsynthesis是这个解码运算中计算量消耗最大的。
- 从这个表可以看出，采用浮点数计算的结果距离我们的实时解码要求相距甚远。



## CCS下实时实现MP3解码

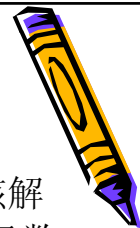
- 对MP3解码程序数据动态范围的统计
  - ◆ 输入输出是16bit，所以最终输出都能用Q15表示。
  - ◆ 通过统计和分析，哈夫曼解码后结果is为整型变量，其绝对值范围不超过512。
  - ◆ 其它主要函数如反量化、混序处理、抗锯齿处理、反DCT变换中间结果最大值为1.2左右，最小值为-1.1左右。
  - ◆ 所以，在Q15格式计算中，基本上拟采用饱和运算来防止溢出，而不用每步都去判断是否存在溢出，并做出定标或扩展精度处理。

## 处理溢出

- DSP的状态寄存器PMST中的SST位可以用来设置饱和运算。
- 当该比特位设置为1时，累加器出现溢出时系统将自动将数据限制在Q15格式能表示的正的最大或负的最小，而不会出现溢出错误。
- 有些函数里面会超出Q15格式的范围比较多。这时，我们会采用重新定标，采用Q11格式防止溢出。

## 用Q15格式实现MP3解码

- 哈夫曼解码函数`III_huffman_decode`: 该解码函数原本就使用整数运算，所以对这个函数可以不做特殊处理，直接将结果数组`is`该为`int`型变量即可。
- 反量化计算函数`III_dequantize_sample`: 从前面的计算量分析表来看，该函数原来采用浮点计算，占用非常多的机器周期，所以这个函数必须使用Q15格式替代原来的浮点技术。



## Q15格式反量化计算函数

- 对反量化公式进行简化，可以得到

$$xr[i] = \text{sign}(is[i]) \times is[i]^{\frac{4}{3}} \times 2^{\text{exp}}$$

- 但在实际的MP3解码数据过程中，`is[i]`和`exp`是有关联的，整个`xr`基本上小于1。所以这个函数完全可以用Q15格式替代。
- 如何使用Q15格式快速计算上面公式？



## Q15格式反量化计算函数

- 对公式上面公式两边取自然对数

$$Y[i] = \ln(|xr[i]|) = \frac{4}{3} \ln(is[i]) + \exp \times \ln(2)$$

- 由于，所以  $|\ln(is)| < 16$ ，因而我们可以使用Q11格式存贮这个中间结果 $\ln(is[i])$ 。
- $is[i]$ 是一个整数，无法直接通过Q15格式的自然对数算法求出，必须将 $is[i]$ 做一定的处理。



## Q15格式反量化计算函数

- 因为 $is[i]$ 为一个整数，设

$$is = C \times (1 - x) \quad \text{且} \quad C = 2^N$$

- 所以 我们可以得到

$$\ln(is[i]) = \ln(C) + \ln(1 - x) = N * \ln 2 + \ln(1 - x)$$



## Q15格式反量化计算函数

- $\ln(1-x)$ 可以通过级数展开求得：

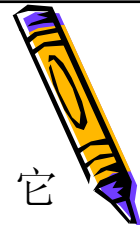
$$\ln(1-x) = -1 - \frac{x^2}{2} - \frac{x^3}{3} - \dots - \frac{x^n}{n}$$

- 这样，我们可以使用Q15格式的自然对数算法求出 $\ln(1-x)$ ，在加上 $N*\ln 2$ ，就完成is[i]的自然对数求解。



## Q15格式反量化计算函数

- SPRA619是一个TI公司的应用报告，它详细给出了自然对数 $\ln$ 的实现代码。
- 其中公式中 $N*\ln 2$ 可以使用查表的方式完成，用Q11格式表示。
- 级数展开公式中的  $1/n$ 用常量表给出，格式Q15。
- 最后完成两个相加，得到Q11格式小数。



## Q15格式反量化计算函数

- 在完成对 $\ln(is[i])$ 的计算后，可以进一步完成 $Y[i]$ 。
- 在公式中，我们需要计算 $\ln(is[i]) \times 4 / 3$ 和 $\exp \times \ln 2$ 。一般情况，除法是较难实现，所以我们用乘倒数的方法来替代除法。这里，我们用乘以0.33333333来代替除3，乘4的操作可以用左移两位来实现。



## Q15格式反量化计算函数

- 在 $Y[i]$ 的计算过程中，我们使用整数（如 $\exp$ ），Q15格式的小数（如公式中的 $x$ ）以及Q11格式的小数，并涉及到它们之间的乘法和加法运算。
- 上面用到的三种格式，其小数点都是固定的，这些运算的关键在于对小数点的处理。
- 对于加法运算，只是需要对齐小数点，便可以方便地使用加法（或减法）指令完成。
- 对于乘法，需要注意的是其结果的小数点位置。

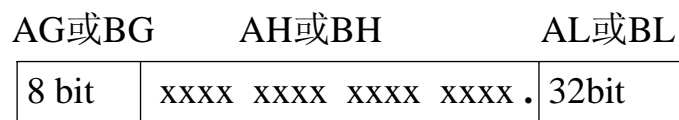


## 多格式定点小数运算

- 该式由两个部分构成： $\ln(is[i]) \times 4 / 3$  和  $\exp \times \ln 2$ 。其中， $\ln(is[i])$  的结果有 Q11 格式的小数表示，乘 0.333333333 (Q15 格式小数) 结果为一个 Q11 格式
- 乘 4 可以使用左移两位实现，我们先不做左移，待完成和第二项的加法时再左移。
- 第二项是一个整数  $\exp$  和一个 Q15 格式小数 0x58b9 ( $\ln 2$ ) 相乘，所以结果也在累加器 A 中，高 16 位表示整数部分，低 16 位为小数部分。

## 多格式定点小数运算

- 右移第一项结果，对齐小数点，完成加法。所以公式的结果  $Y[i]$  被存放在累加器 A 中，高 16 表示整数部分，低 16 位表示小数部分。



整数与 Q15 格式小数乘积的小数点位置



## Q15格式反量化计算函数

- 在求得 $Y[i]$ 后，我们可以利用指数计算出中的反量化结果。其指数计算可以通过下面的级数展开求得：

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

- 所以，

$$Xr[i] = e^{Y[i]} = e^{lk} \times e^x$$

## Q15格式反量化计算函数

- 级数展开使用**POLY**指令实现。
- SPRA619**也给出了指数计算的详细代码。
- 其中整数部分的指数计算使用一个常数表**exptbl**，通过查表求出（使用整数来近似）。
- 另外还定义了一个常量表**a9**来表示 $1/n!$ ，将除法变为乘法运算，最后使用Q15格式保存反量化的结果值。

## Q15格式反量化计算函数

- 反量化函数使用**Q15**格式后需要的机器周期（单次调用平均数）从浮点版本的**5204569**个减少到**167088**个，整个计算量只有原来的**3.2%**左右。
- 可以看出这里的计算是一个速度和精度的折中。



## 用Q15格式实现MP3解码

- 混序处理函数**III\_reorder**占用时间消耗不多，所以没有做特别处理，仅启用优化编译开关后也能达到较满意的结果。
- 抗锯齿处理函数**III\_antialias**直接在**C**代码上修改，使用**Q15**格式替换原来的浮点计算。



## Q15格式抗锯齿处理函数

- 主要修改有两项：先将原来的两个数组 **ca**和**cs**直接使用**Q15**格式定义出来
- 然后使用**32**位长整型做整数乘法，将结果左移一位，保存常整型变量的高**16**位。

```
// C语言长整型整数乘法实现Q15格式小数乘  
q15_1 = ((long)(bu * cs[ss]) - (long)(bd *  
ca[ss])) << 2;
```

```
.....  
// ---- 结果在高16位中  
hybridIn1[sb][17-ss] = q15_1 >> 16;
```

## Q15格式抗锯齿处理函数

- 函数**III\_antialias**每次调用的机器周期数从原来的**120729**下降到**17909**，约为原来的**15%**。
- 函数**III\_hybrid**的主要功能是计算反DCT，所以在这个函数中，我们对IMDCT做了特别处理

## Q15格式反DCT变换

- 反DCT采用下面的公式计算

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

- 为了方便Q15格式的计算，我们使用查表法求得COS值。根据浮点版本的反DCT计算函数inv\_mdct，我们将两个COS的计算预先算出，并存放在数组COS\_1和COS\_2中。同时将另外一个窗口系数也用数组win保存。



## Q15格式反DCT变换

- 这个整数版的inv\_mdct函数不能直接使用，主要问题在于编译完成后，整数计算只保存累加器的低16位，而Q15格式的小数计算需要保存累加器的高16位。
- 所以我们需要将这个C代码编译成汇编代码，并手工修改。其方法是将MPY乘法指令和MAC乘累加指令后的存贮操作改为STH，即保存累加器的高16位。



## Q15格式反DCT变换

- 另外，还需要设置FRCT小数乘法状态位以及PMST寄存器的SST饱和运算状态位，并根据具体应用决定使用RET或FRET作为返回指令。
- 使用修改后的汇编的效率得到大大提高，单次调用III\_hybrid函数的机器周期从原来浮点版本的189468，下降到1747个（其中反DCT函数inv\_mdct消耗了1304个机器周期），只有原来的1%。



## 子带合成滤波函数 SubBandSynthesis的修改

- 这个函数的处理方法和前面的inv\_mdct处理方法一致，也是先改为使用整数变量的C代码，使用优化C编译器编译产生汇编源程序，最后手工修改汇编源程序。
- 该函数的机器周期消耗从浮点版本的595742，下降为24424，只有原来的4%。



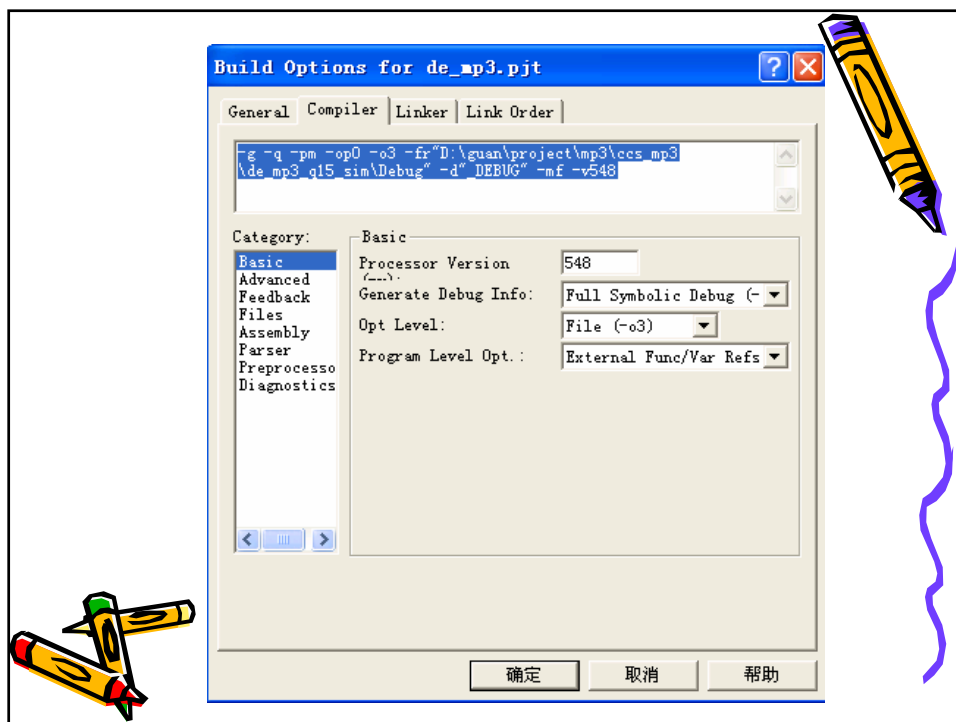
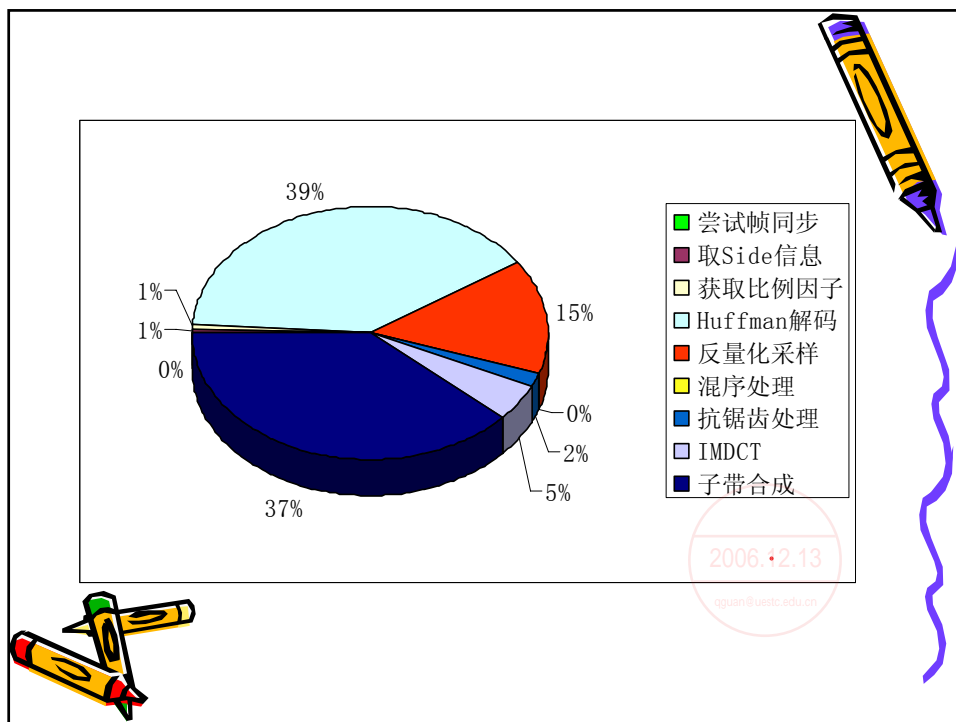
## 采用Q15格式后主要函数 消耗的机器周期



- 单通道每帧MP3解码其主要函数的总的机器周期（单次调用的机器周期乘以调用次数）消耗约为2301766，若在100MHz的C5400上运算，每个机器周期为10ns，所以每帧数据需要约23ms。
- 而从前面的分析我们了解到，MP3播放时每帧数据的时间间隔为26.122ms，所以我们认为这个Q15格式的版本可以实施实现单通道的解码播放。



函数名	单次调用机器周期个数	每帧数据调用次数（单声道）	功能	
Seek_sync	415	1	尝试帧同步	
III_get_side_info	12730	1	取Side信息	
III_get_scale_factors	3901	3	获取比例因子	
III_huffman_decode	303798	3	Huffman解码	
III_dequantize_sample	167088	2	反量化采样	
III_stereo			立体声处理	没有使用
III_reorder	2229	2	混序处理	
III_antialias	17909	2	抗锯齿处理	
III_hybrid	1747	64	IMDCT	
SubBandSynthesis	24424	36	子带合成	
总的机器周期数			2301766	



## 总结

- 不能使用浮点数版本在定点DSP上运算（仿真运算），但可以在前期调试以及探针工具。
- Q15格式的定点数版本运算
- 小数点位置的灵活应用以及对算法的改进
- 对溢出的控制处理
- 查表方法的应用（用空间换时间）
- 对C代码的修改或汇编改写代码
- 优化编译选项的应用
- 内存优化



## 思考题

- 与浮点解码比较，Q15定点解码的误码率和SNR。
- 对huffman解码函数的优化？
- 对MP3原始数据读入操作的优化？
- 实现立体声解码的实时实现？

