# 解析wav文件

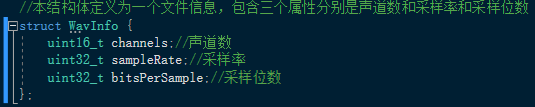
在进入opus主函数的时候，需要将wav文件携带的音频数据解析出来，在git下载下来的程序中自带了wav解析函数



解析过程参考

[WAV格式文件分析\_wav格式解析-CSDN博客](https://blog.csdn.net/weixin_48680270/article/details/123523517)

大概过程是根据wav的不同格式块去判断最后数据块的位置，从而读取出音频数据，最终解析成raw音频数据流，同时读取格式头保存wav文件中音频格式有声道数，采样率，采样深度，总样本数。opus中新建了一个结构体用来存储音频格式：



后面会用采样率和声道数计算编码结构体的初值

# 创建编码结构体

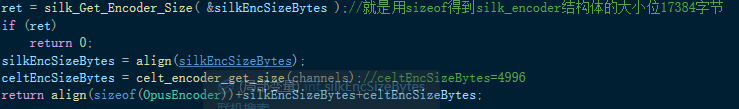


OpusEncoder结构体和创建编码结构体的函数都是opus编码函数库中固定的编码结构体，这个结构体主要保存了编码过程的初值和过程值，传入值分别为采样率，声道数，opus应用模式和记录错误的参数；

首先为结构体分配内存：

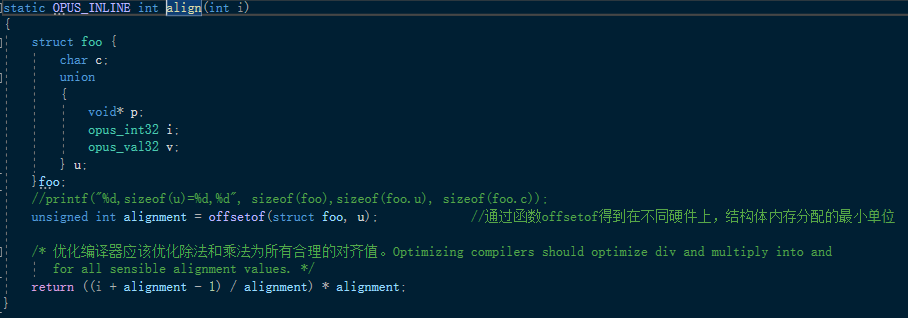


为了得到在不同硬件上，结构体内存分配的最小单位，这里使用了内存对齐：



例如，‌对于64位系统，‌每次内存访问通常是以8字节为单位进行的，‌这样可以更有效地利用硬件性能。‌如果数据没有按照对齐规则存储，‌那么在访问这些数据时，‌CPU可能需要多次内存访问才能完成一次操作，‌这不仅增加了CPU的工作量，‌还降低了数据处理的效率。‌因此，‌通过合理的内存对齐，‌可以高效地利用硬件资源，‌提升程序的执行效率。‌

其中align函数为：



通过函数offsetof得到不同硬件上结构体内存分配的最小单位，在64为计算机系统中alignment为8，也就意味着这个结构体的内存大小应该为8的整数倍，在return语句中就是为了调整为alignment的整数倍。

之后初始化编码结构体：



用到了采样率和声道数，及opus的应用模式。

编码器内部结构由opus状态，silk状态，celt状态组成，下面是编码器内存示意图

|  |  |  |
| --- | --- | --- |
| Opus状态编码器占2208字节 | Silk编码器占17384字节 | Celt编码器占4996字节 |

# 初始化编码器

opus结构体变量及解释：

silk结构体变量及解释：

celt结构体变量及解释：

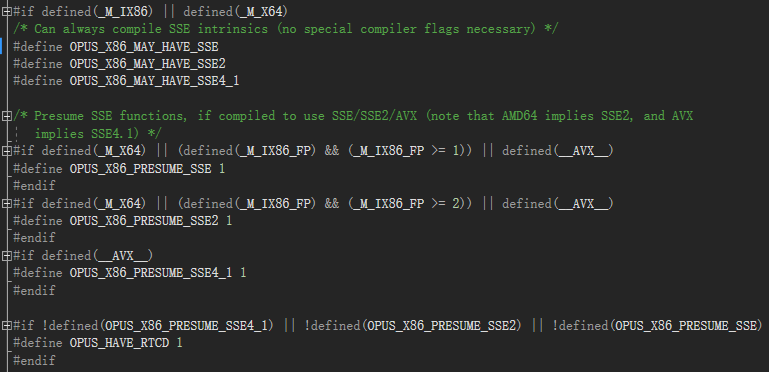
下面是初始化的详细过程：

## 确定硬件或软件架构

通过不同架构的宏定义来确定采用定点的函数还是浮点的函数



这个可能在不同的架构上有不同的参数定义，详见opus中的config.h文件：



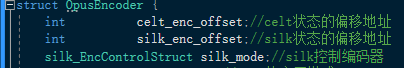
## 初始化opus和silk编码器状态

**先初始化opus部分状态：**

初始化流的声道数=opus编码器声道数=音频声道数；

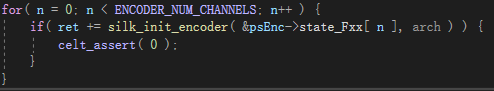
opus编码器采样率=音频采样率；

接下来初始化silk状态（state）和silk控制（status）编码器，这里是两个不同的结构体，后面的是控制编码器位于opus结构体中的silk\_mode结构体，是opus控制silk的，前者是silk独立的编码器状态结构体，为了区分，前者叫silk状态（编码）器，后者叫silk控制（编码）器。silk控制（编码）器位于opus结构体中的位置为：

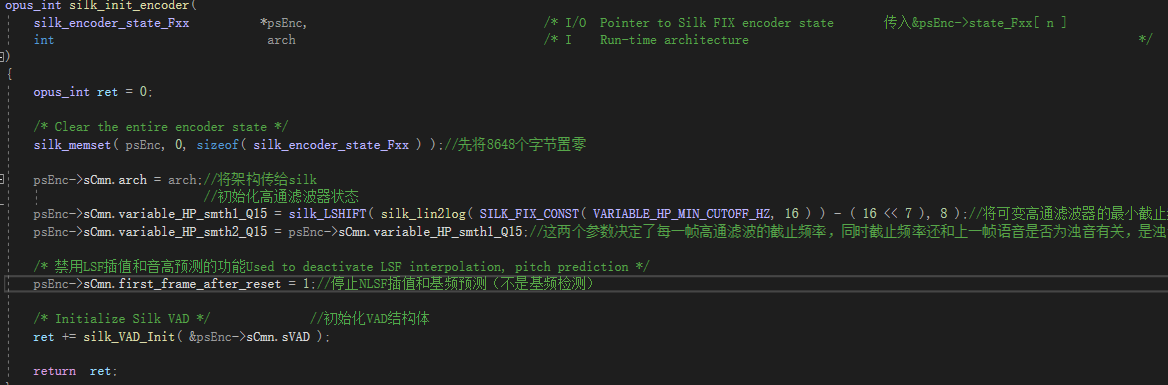


**然后中间插入重置silk状态器:**

方式是全部置零，然后赋初值，这里根据声道数来初始化状态器结构体中结构体数组的个数，最大声道数为2：



函数为：



具体过程为：

将架构值arch传给silk；

初始化高通滤波器两个状态参数，这两个参数决定了每一帧高通滤波的截止频率，同时截止频率还和上一帧语音是否为浊音有关，是浊音则需要重新计算这两个参数，不是浊音则不需要重新计算；

停止NLSF插值和基频预测（不是基频检测）；

初始化silk状态器中的VAD结构体：

VAD初始化函数：

* 初始化了噪声等级的基值也就是默认背景噪声是粉红噪声（以噪声等级偏移的变量存在）
* 初始化了噪声等级的倒数
* 初始化了帧计数器初值设置为15，在计算平滑噪声的时候需要用到帧计数器记录VAD处理的帧数来计算平滑噪声的参数，保证前1000帧（20s）参数最大，使噪声平滑的很快
* 初始化了每个子带的能量噪声比值默认20db也就是信号能量是噪声的100倍

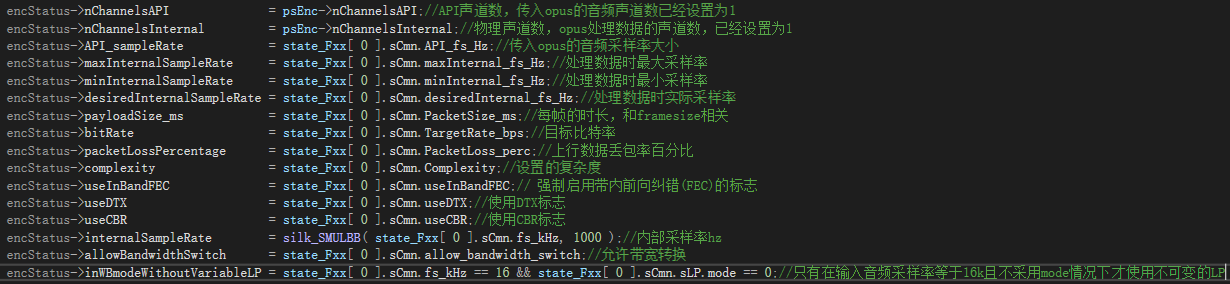
上面提到的粉红噪声又称为1/ƒ噪音（有时也称作闪烁噪声）是一个具有功率谱密度（能量或功率每赫兹）与频率成反比特征频谱的信号或过程。在粉红噪声中，每个倍频程（也就是对数坐标）内都是等量的噪声功率。

**回归opus初始化：**

函数间传递音频数据采用单声道，函数内处理数据也采用单声道，这使得opus强制输出单声道压缩参数，解压缩就一定是单声道音频。

**初始化opus中的silk控制器：**

将silk状态器的部分状态（下图）赋值给控制器,这里只有API和物理声道数有值其余为0，赋值给silk控制器，这里**可能**是为了保证初始化的时候silk状态器和控制器的部分状态一致；



然后设置默认值，更改opus默认参数可以从初始化函数里直接更改，也可以通过可变参数函数来实现赋值：

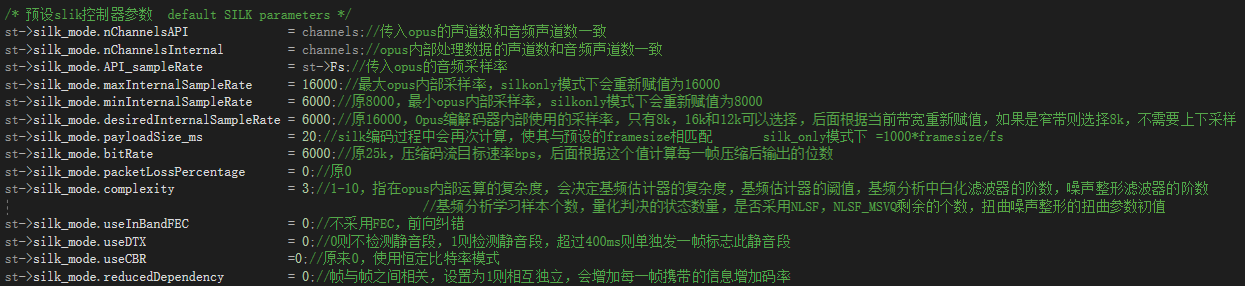


例如：

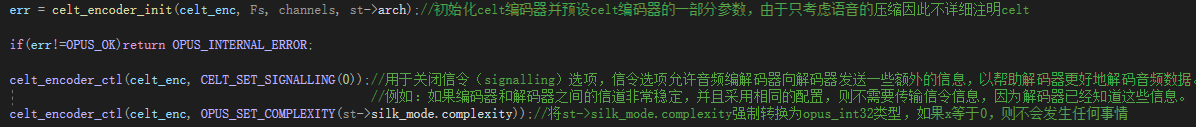
是将celt\_enc结构体中signalling变量赋值为0；函数第一个参数是可变的结构体类型是OpusCustomEncoder\*在例子中就是celt\_enc，第二个参数需要通过列表去确认变量，列表是由函数映射实现的，在例子中就是CELT\_SET\_SIGNALLING，体现为不同名称的函数，其中映射值就是最后要赋值的值，在例子中就是0.

**设置silk控制器默认参数：**

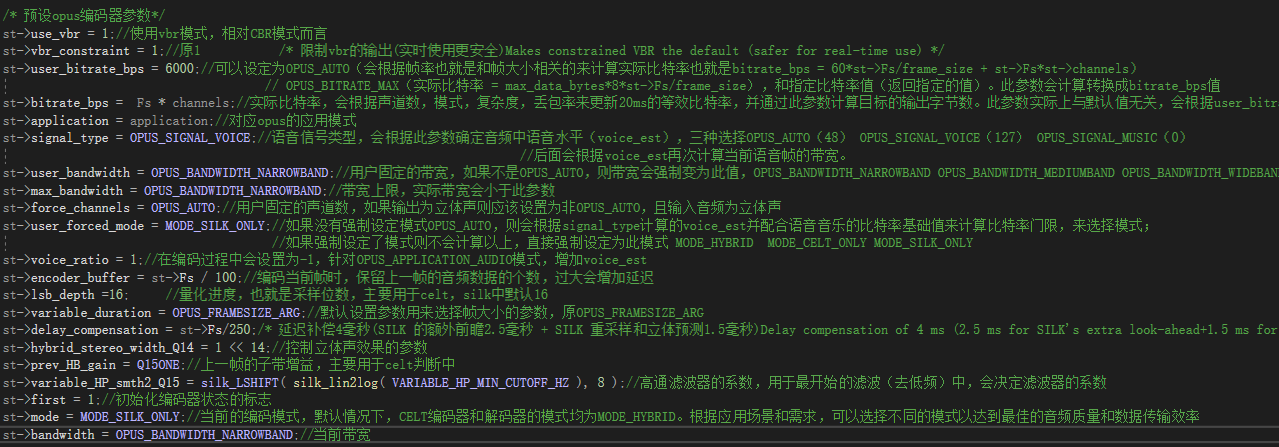
预设了以下参数：



**初始化opus中的celt控制器：**



**初始化opus参数：**



至此完成初始化。

后面就是开始使用此结构体进行编码音频，每次压缩一帧输出的参数由cbits数组记录并输出，个数由nbBytes记录，固定默认的输出模式是先输出当前参数的个数nbBytes（四字节），然后再输出cbits数组参数（每个参数都是一字节）

## 编码前

确定一帧最大输出字节，一般是固定的3\*1276

如果应用模式采用低延迟模式则延迟buffer长度设置为0，一般来说会有4ms的上一帧数据加入当前帧数据中一起计算，采用低延迟应用模式则没有这个数据

计算立体声宽度，单声道音频为0

buffer长度=延迟buffer=st->Fs/250，也就是4ms，在初始化中就已经计算完成

将用户设定的比特率计算为实际比特率，除非 user\_bitrate为OPUS\_AUTO或者OPUS\_BITRATE\_MAX，否则st->bitrate\_bps=user\_bitrate

计算帧率，取决于设置的帧大小和当前音频的采样率，1s中有几帧

如果采用CBR，更新cbr的输出字节，同时根据此字节数更新实际比特率

判断最大字节是否符合标准，最大字节小于3或者一帧至少输出三个字节，或者帧率小于50（也就是一帧小于20ms）时全部满输出还小于300字节也就是比特率小于2400空间太小发出“ PLC”帧

# 日志

20240815

将之前的疑惑转移到这里，opus中为什么基频分析的码本指的是矢量量化？可能需要学习矢量量化的原理；

在压缩基频信息的时候为什么一个码本索引就可以传递出基于中心值的偏移？

噪声整形中扭曲参数是什么，作用是什么，决定噪声谱形状？如何起作用的？