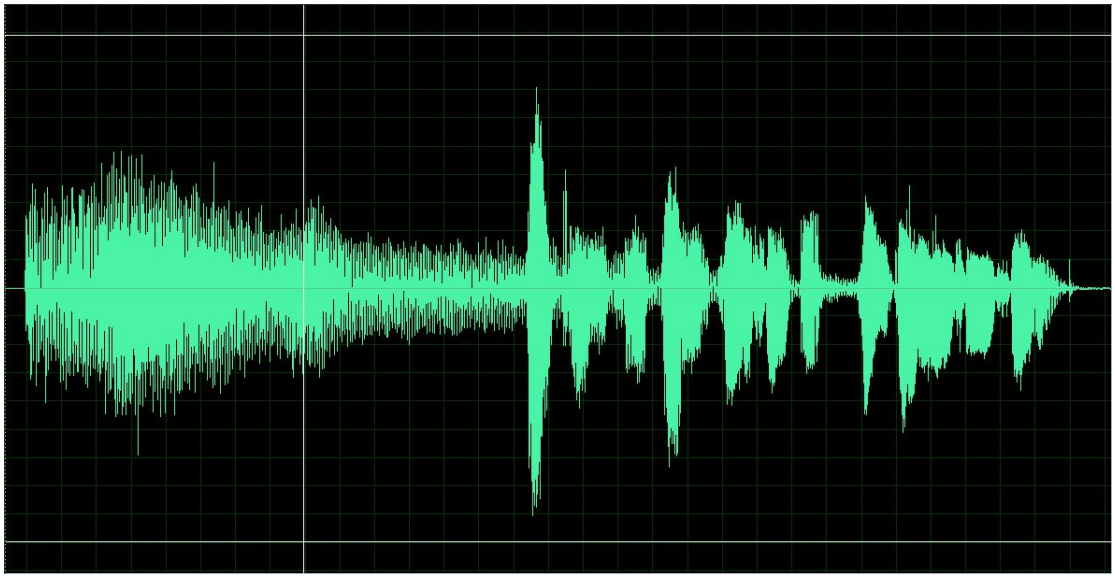
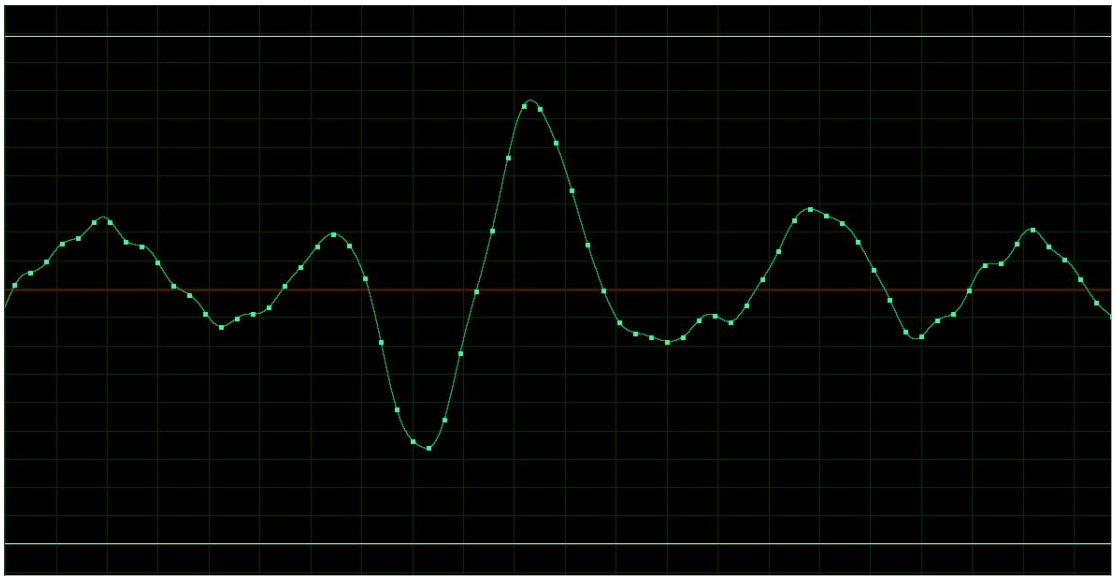
PWM 语音播放

先来看看音频是什么样子，这是一段音频数据的波形



下面是放大的局部图：



一定的速度采样（ADC）这些波形进行存储，就是音频数据了，所以播放就是按原来采样的速率再用 DAC 输出对应的数据即可。
这里的音频有两个主要的参数，采样速率和采样位数，

采样速率：指 1s 中采样多少个数据点，比如 1s 种采集 16000 个点，那么采样率就是 16KHz。采样速率越高，越能抓到频率较高的声音，比如 CD 的采样率就是 44.1KHz，确保人耳能听到的声音都会被抓到。

采样位数：是指音频幅度最大值与最小值分为了多少阶，比如满幅度是 3.3V，如果是 8Bit 位数，那么每一阶就是 3.3V/256 = 12.89mv，采样位数越高，声音细节越好，所以采样速率和位数越高，声音还原越逼真，但存储的数据量也越大，一首三四分钟的歌曲，如果不采用编码按原始波形数据存储，数据量有好几十兆大小，这涉及到音频编码的问题，这里不展开

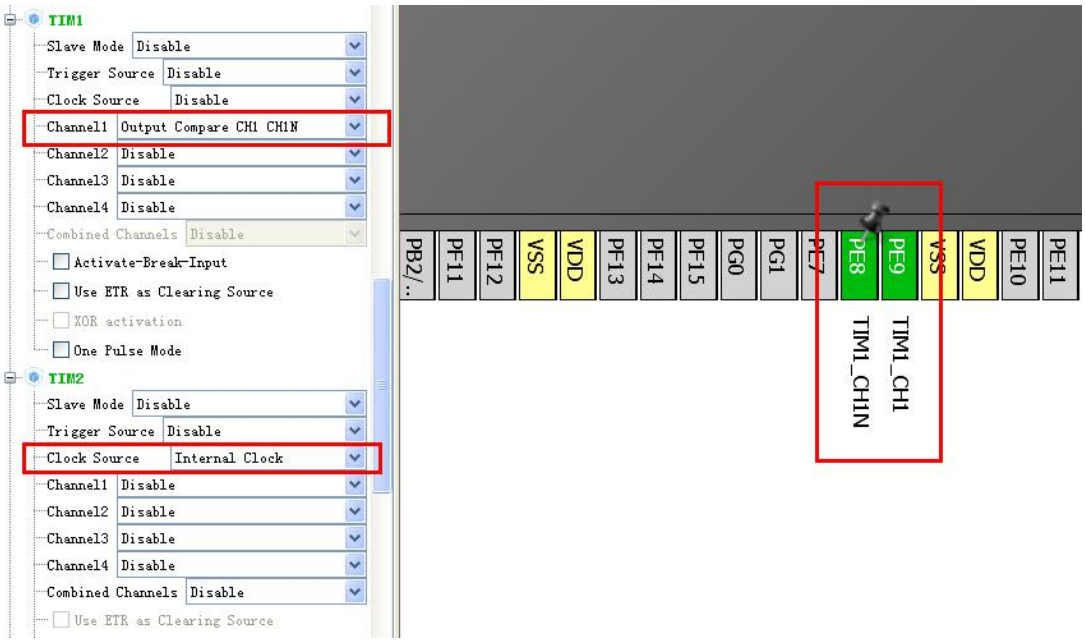
讲了，有兴趣的同学可以找相关资料。

接下来看怎么播放，最简单的当然是把采样（ADC）的数据按原样输出（DAC）了。但我们有些芯片本身不带有 DAC，所以只能用 PWM 代替 DAC，PWM 即脉冲宽度调制。这里只需要把 DAC 的幅度值转换成 PWM 的占空比即可，例如 16KHz 8Bit 的声音转换成 16Khz 256 阶占空比的 PWM。但有一个问题，如果用 16KHz 的 PWM 播放语音，声音是可以播放，但有一个 16Khz 的谐波存在，这个声音会被人耳听到，所以需要更高频率的 PWM，数据还是按照 16Khz 更新。

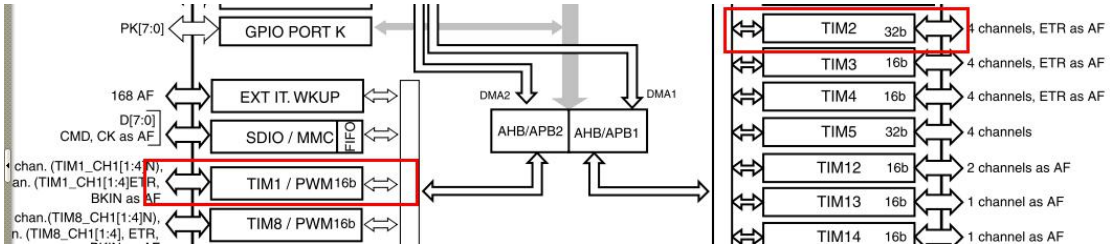
我这里使用 32KHz 的 PWM，用 16KHz 8Bit PCM 格式的音频数据，8Bit 的数据对应一个 Byte，16KHz 采样，1 秒种占用存储空间就是 16K Byte，因为 F429 有 2M Byte 的 Flash 存储空间，理论上可以存储 2048K/16K = 128 秒的音频。下面是用 NucleoF429 实现音频播放的具体过程：

一、配置 PWM

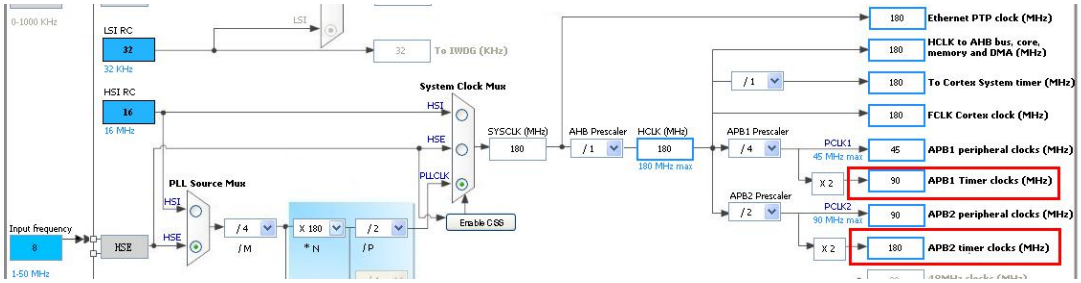
1、用 STM32CubeMx 建立工程，配置两个定时器 TIM1 和 TIM2，TIM 用于 PWM 产生，TIM2 用于 16KHz 数据更新，



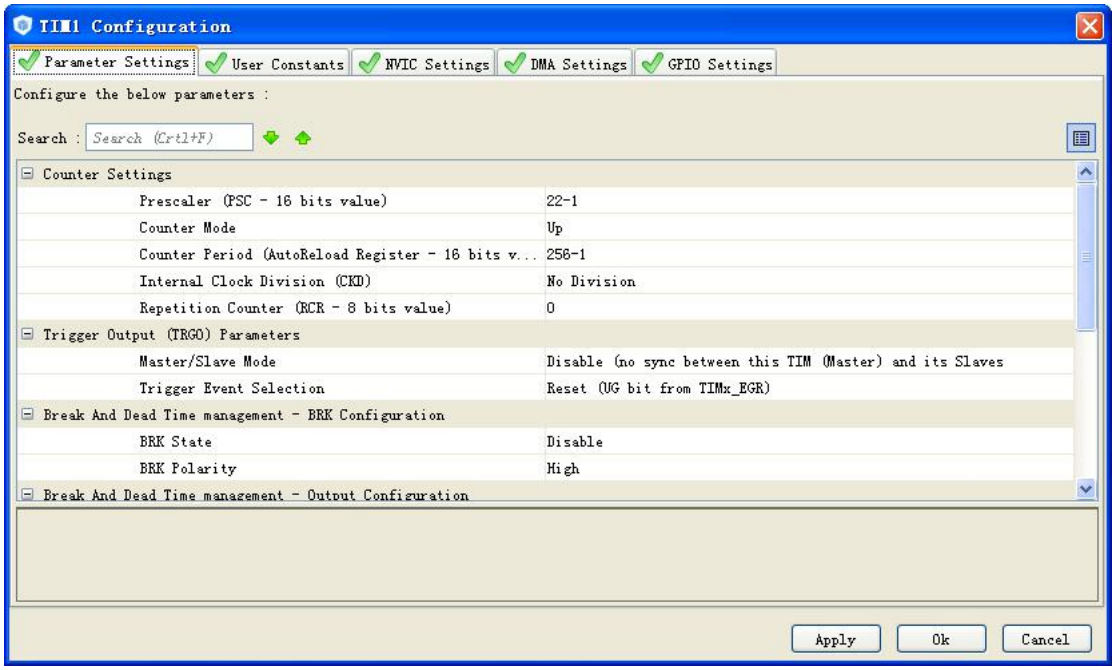
TIM1 选择 PWM 互补输出（单通道也可以），将 PE8 和 PE9 复用为 PWMN 和 PWMP。
TIM1 在 APB2 总线上，TIM2 在 APB1 总线上



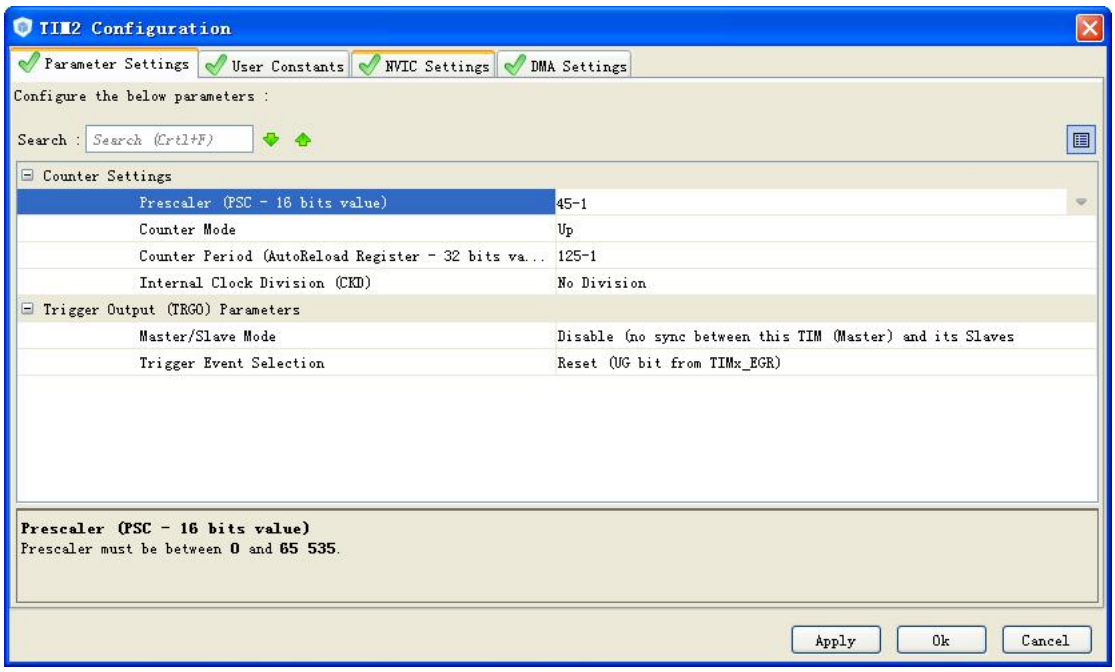
所以 TIM1 和 TIM2 的时钟频率分别为 180M 和 90M，系统时钟用 HSE 输入的 8MHz。



将 TIM1 设置为 32KHz，即 31.25us。8Bit 占空比，一个 LSB 为 31.25us / 256 = 0.1220703125us = 8.192MHz，TIM1 180M / 8.192M = 21.97265625，这里取整数 22。所以实际的 PWM 频率为 1/（180 / 22） *256 = 31.289us = 31.96KHz



TIM2 为 90MHz，45 分频后为 2MHz 即 0.5us，周期 125 即 62.5us = 16KHz。



NVIC 开启 TIM2 中断。生成工程名和目录后生成 Keil 工程。

二、播放语音

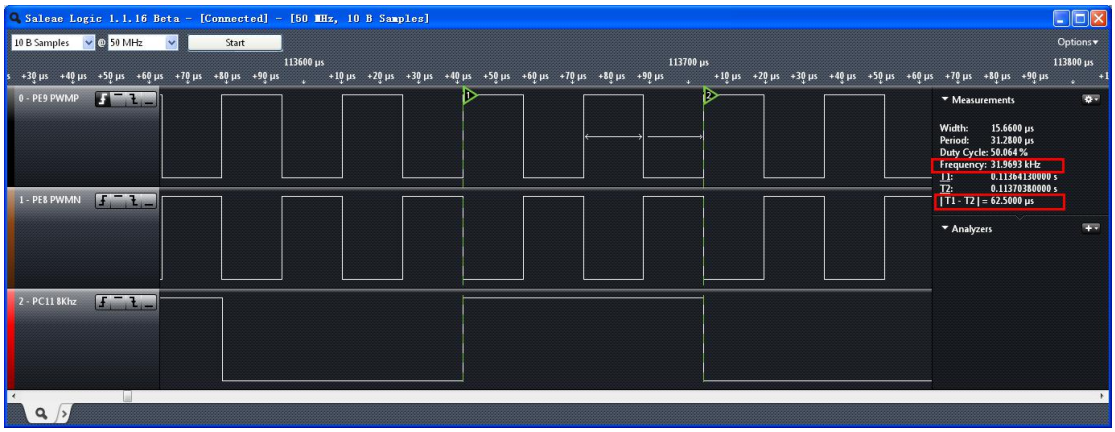
1、先编译后，编写 TIM 中断服务程序。

```
24  void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
25  {
26      switch ((uint32_t)(htim->Instance))
27      {
28      case (uint32_t)TIM1:
29      break;
30
31      case (uint32_t)TIM2:
32      HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_11);
33      break;
34
35      default:    break;
36      }
37
38  }
39
```

完成后，开启 TIM2 中断和 PWM，（PWM 是互补输出，需要单独开启各个通道）

```
84    /* Initialize all configured peripherals */
85      MX_GPIO_Init();
86      MX_TIM1_Init();
87      MX_TIM2_Init();
88      HAL_TIM_Base_Start_IT(&htim2);
89      HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
90      HAL_TIMEx_PWMN_Start(&htim1, TIM_CHANNEL_1);
91
```

用逻辑分析仪测量输出波形。



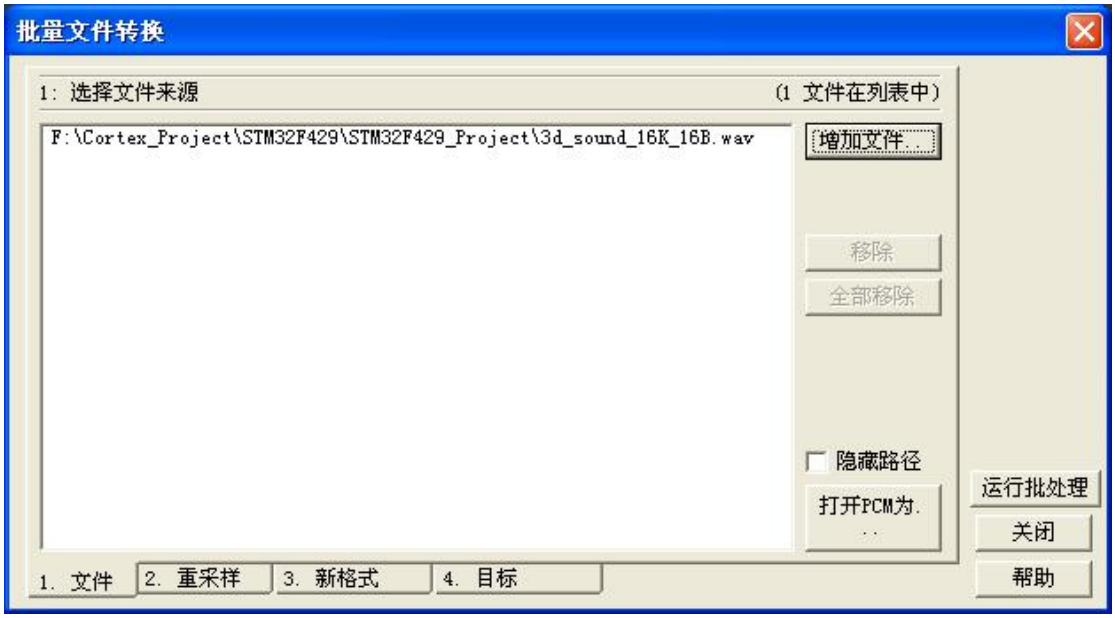
如图所示，TIM1 PWM 为 31.96KHz，TIM2 为 62.5us 即 16KHz，结果正确。

接下来处理音频：

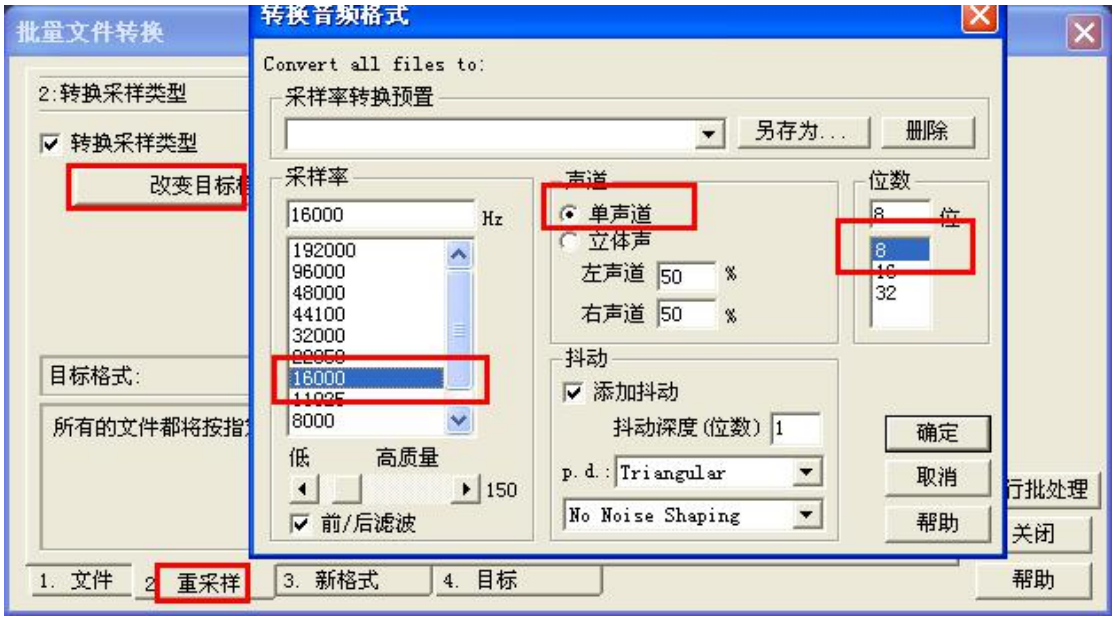
这里使用的音频是 PCM 格式，是未进行压缩编码的原始数据，可以直接给 PWM 输出。

音频处理的软件有许多，只要能把格式转为 PCM 即可，下面是我用 Cool Edit 这款软件做的音频格式转换。

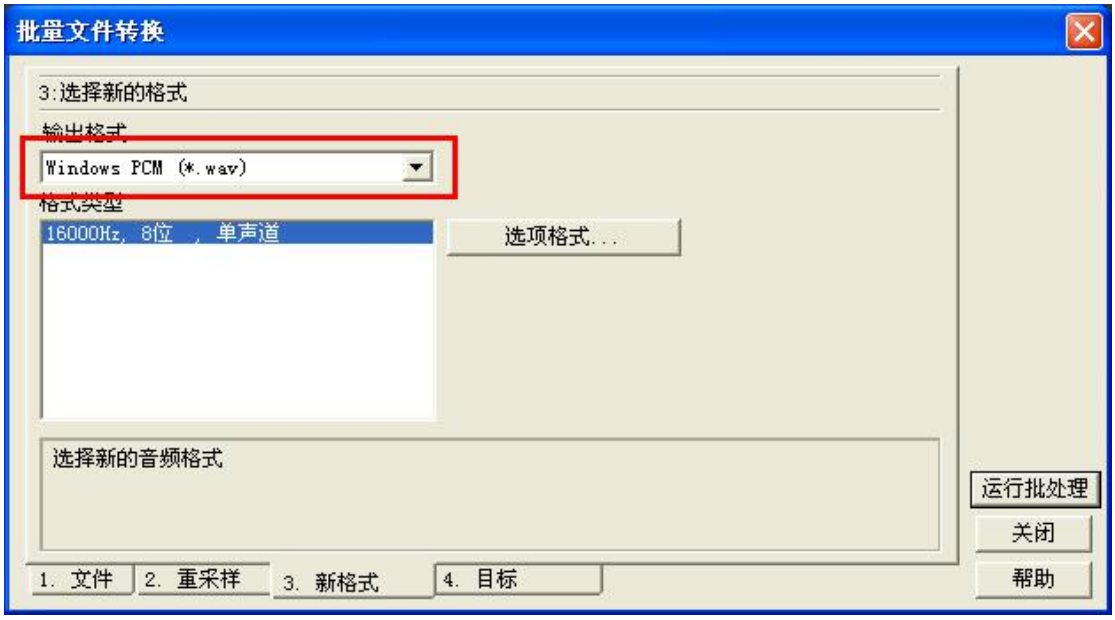
选择菜单 文件-->批量转换



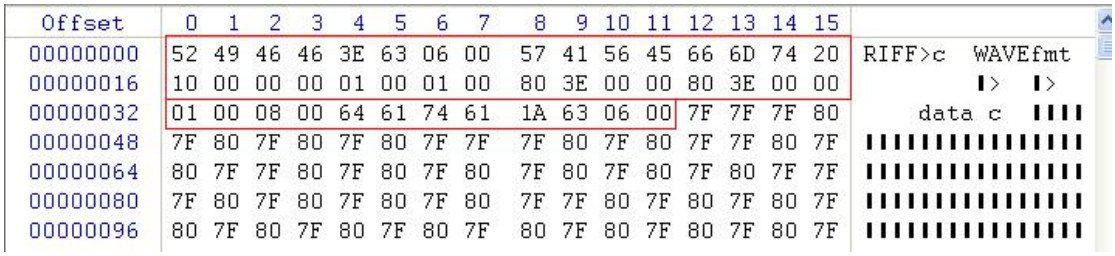
选择新的采样率和采样位数。



选择 PCM 格式。设置输出目录后运行批处理完成转换。

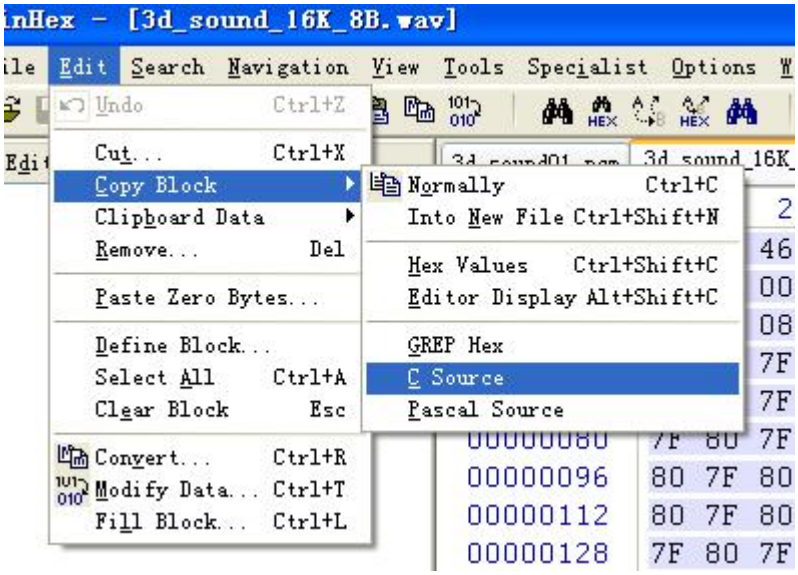


完成后的音频文件用 WinHeX 这个软件打开。



Offset	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
00000000	52	49	46	46	3E	63	06	00	57	41	56	45	66	6D	74	20
00000016	10	00	00	00	01	00	01	00	80	3E	00	00	80	3E	00	00
00000032	01	00	08	00	64	61	74	61	1A	63	06	00	7F	7F	7F	80
00000048	7F	80	7F	80	7F	80	7F	7F	7F	80	7F	80	7F	7F	80	7F
00000064	80	7F	7F	80	7F	80	7F	80	7F	80	7F	80	7F	80	7F	7F
00000080	7F	80	7F	80	7F	80	7F	80	7F	7F	80	7F	80	7F	80	7F
00000096	80	7F	80	7F	80	7F	80	7F	80	7F	80	7F	80	7F	80	7F

图中红框中的 44 个 Byte 为 PCM 格式的文件头信息，后面的的数据为音频数据，数据全选后利用 WinHex 的可选格式复制



将数据以 C 数组的形式导出，在工程目录下新建.h 文件，将复制的文件粘帖到.H 文件并在工程中 Include 进来，定义起始和结束地址，数组的大小即为文件结束地址，数组用 const 修饰，可以将数据存储到 Flash 中。

```
#define AudioDataStartAddr  44
#define AudioDataEndAddr    418630
const unsigned Audio_data[418630] = {
    0x52, 0x49, 0x46, 0x46, 0x3E, 0x63, 0x0
```

在 TIM2 中，以 16KHz 的速度更新 PWM 数据即可实现音频播放。

```
32      case (uint32_t)TIM2:
33      //HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_11);
34      if(DataCnt++ >= AudioDataEndAddr)
35      {
36          DataCnt = AudioDataStartAddr;
37      }
38      PWM_SetData(TIM1,Audio_data[DataCnt]);
39      break;
```

编译工程，下载到 NucleoF429 板子上，在 PE8 或 PE9 上接一个喇叭即可听到声音。

以上用的音频采样是 16K 8Bit，要想提高音质，提高采样和 Bit 数即可，音量可以用外接三极管或功放放大，音频数据也可以用 ADC 采集后存储到 SPI Flash 后播放，实现录音回放。

----此文来自 ARM 中文社区