2020 年江苏省大学生电子设计竞赛

放大器非线性失真研究装置（E 题）

【本科组】

2020 年 10 月 13 日

目录

摘　要

根据集成运放电路的设计原则制作出了一款放大器非线性失真研究装置，能够生成放大后的标准正弦波，以及四种失真的正弦波形（顶部失真、底部失真、双向失真、交越失真），并计算出任意波形的总谐波失真。

一、设计方案工作原理

1.1 预期实现目标定位

使用晶体管、阻容元件等元器件制作一个受控晶体管放大器，并用其研究信号的几种常见非线性失真（顶部失真、底部失真、双向失真、交越失真）。晶体管放大器预期具有输出“顶部失真”、“底部失真”、“双向失真”、“交越失真”的电压波形。能够通过单片机计算出每种失真的“总谐波失真”。

1.2 技术方案比较

1.2.1 输出正常放大的正弦波以及各种失真的晶体管电路设计

根据赛题要求，需要将输入的频率 1kHz、峰峰值 20mV 的正弦波放大为峰峰值 2V 以上的正弦波，即将输入放大 100 倍以上。为实现这个要求，我们需要使晶体管制作一个放大电路。我们在查阅资料后定下方案，参考集成运放的设计方法进行电路设计，放大器采用输入级——中间级——输出级的设计思路。

a.输入级设计

首先，集成运放的输入级一般是信号决定信号是否良好的关键因素，要求其输入电阻高、电压放大倍数大、抑制零点漂移能力强、静态电流小。

在集成电路中经常使用差分放大电路作为输入级，差分放大电路以两只管子集电极作为输出，能够很好的克服温漂，但是差分放大电路的结构相对复杂，要调节的参数较多。

还可以使用经典的阻容耦合的共射放大电路作为输入级，这种方式抑制温漂的能力不如差分放大电路，但是需要调节的参数较少，电路相对简单。

由于比赛时间仅有四天三夜，除去各种波形的产生，我们还要处理失真波形，分离出基波和谐波，工作量较大，而且题目对放大倍数没有特别高的要求，所以我们选择使用相对比较简单的阻容耦合的共射放大电路作为输入级。

b.中间级设计

中间级是集成运放的主放大器，一般会使用复合管作为放大器，因为放大倍数较小的缘故，只需要单管即可满足要求。

c.输出级设计

集成运放的输出级一般应具有电压线性范围宽、输出电阻小、非线性失真小的特点。为满足这些要求，我们决定采用互补输出电路。

1.2.2 五种输出电压的“总谐波失真”的测量电路设计

对于测量总谐波失真这个要求，查阅资料可以得到总谐波失真（THD）的计算公式为

$$\mathrm{THD}=\frac{\sqrt{U_{o2}^2+U_{o3}^2+U_{o4}^2+\mathrm{L}+U_{on}^2}}{U_{o1}}\times 100\%$$

由公式可知，如果要求得 THD，就需要先得到基波和谐波分量。那么我们此题的难点就在于如何得到这个基波和谐波分量，为了实现这个目标，我们结合已经学过的知识和查阅资料，得出了两

种方案：

a.通过硬件滤波分离基波和谐波

硬件处理失真波形一般可以采用双 T 型带组陷波电路等电路完成基波和谐波的分离。

b.用 STM32 单片机控制波形产生，借助 FFT 进行频谱分析

除了硬件滤波外,还可以通过对失真波形进行采样，得到采样序列，然后对采样序列做块速傅里叶变换，进行频谱分析，提取基波和谐波分量，计算出总谐波失真。STM32 也有相应的 DSP 库，可以完成对快速傅里叶变换的计算。

c.用 FPGA 控制波形产生，借助 FFT 进行频谱分析

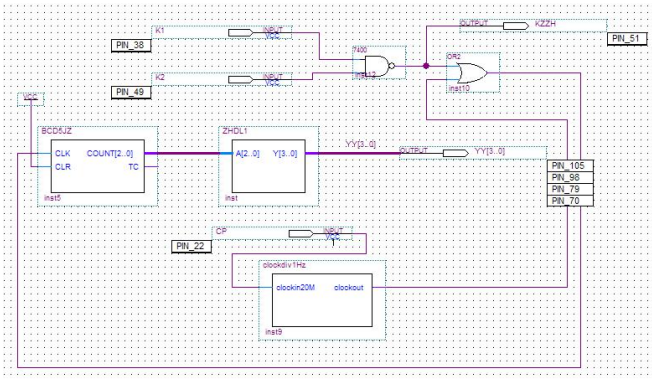

图 1.2.1　FPGA 控制波形的实现

我们结合自身的实际情况，组内成员更加擅长软件设计，所以选用软件的方式实现 THD 的测量计算，我们也尝试用 FPGA 做过控制波形的产生，但最终因为 FPGA 的准备不够充分而没有选择该方案。而选择采用 STM32 完成电路控制以及对频谱的分析。

1.3 系统结构工作原理

向输入端输入一个频率为 1kHz，峰峰值为 20mV 的正弦波，在输入级处设置一组分压电阻，用于对输入信号进行衰减，调低该阻值来使输出发生双向失真，在输入级调节电阻阻值来调整静态工作点，用于实现正常输出波形、顶部失真、底部失真。在输出端使用的是有源互补输出电路，由于只有当输入电压大于三极管的导通电压时输出电压才会随着输入电压变化,要消除交越失真就要使输入电压在工作死区内，通过增加死区范围就能够消除交越失真，反之，将二极管短路就可以实现交越失真。

图 1.3.1 - 晶体管放大电路电路流程图

1.4 功能指标实现方法

1.4.1 放大器能够输出无明显失真的正弦电压

输入电压在进入输入级电路前先经过一个分压电路进行信号衰减，通过一个模拟开关控制是否进行衰减，然后通过一个阻容耦合的 PNP 共射放大电路，通过调整 R7、R8 的阻值来调整静态工作点，再经过一个三极管二次放大后电压达到指定要求后经过互补输出电路后输出放大到 2V 以上的标准正弦波。

1.4.2 放大级能够输出有顶部、底部失真的电压

阻容耦合的 PNP 放大电路中，通过调节 R7、R8 来调整静态工作点，以实现顶部失真和底部失真。用模拟开关来控制这两组不同阻值的电阻的接通。

1.4.3 放大器能够输出有双向失真的电压

通过调整输入放大电路前是否对信号进行衰减来决定输出是否失真。由于晶体管放大电路中的电阻参数都是在信号被衰减后得出的，所以在去除这个衰减后输出幅度会大幅增大导致顶部和底部同时失真。

1.4.4 放大器能够输出有交越失真的电压

在互补输出级处加了两个二极管电路用于增大死区以消除交越失真，将这两个二极管电路短路后就能得到原本的交越失真波形。

1.4.5 分别测量并显示五种失真波形的总谐波失真

使用 STM32 单片机对输出波形进行采样，然后将采样序列进行快速傅里叶变换,得出频谱，再从频谱中提取基波和各次谐波分量，计算出总谐波失真。

1.5 测量分析处理

计算得出静态工作点，标定正常输出正弦波时的静态工作点，再通过 Multisim 仿真大致确定 R7、R8 的值，连接完电路实测，观察波形调节电位器。

二、核心部件电路设计

2.1 电路结构工作机理

本系统由晶体管放大器、波形的切换控制、电压采集以 THD 的计算三部分组成。

2.1.1 晶体管放大器

本部分可以细分为电源处理电路、信号衰减电路、晶体管放大电路。

a.电源处理电路

对于一个电路系统来说，一般有多个负载，这些负载的供电都来自于同一个电源，每个负载上的电压在不停地变化，这就导致 Vcc 供电处的电压出现很多的波纹噪声，如下图所示

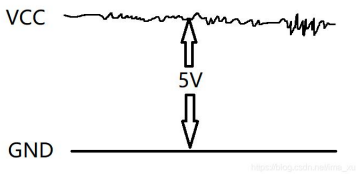

图 2.1.1 - 波纹噪声

为了尽量减小供电不稳给电路带来的影响，我们就需要在每个模块的电源供电处接上一个 100uf 以上的铝电解电容作为去耦电容，同时也可以滤去高频噪声，本模块共使用两个 100uf 的电解电容作为去耦电容。

b. 信号抑制电路

将通过电阻分压电路将输入的正弦信号衰减掉一部分用于放大器的输入，设计相应的晶体管放大电路，使得衰减后的信号输入到晶体管放大电路中可以产生正常的正弦波，这样调节两个分压电阻的阻值，就可以使输出幅度过大导致顶部底部同时失真。电路如下图所示：

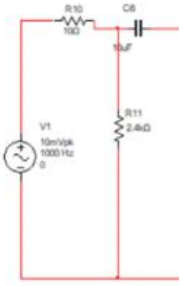

图 2.1.2 - 信号衰减电路

c.晶体管放大电路

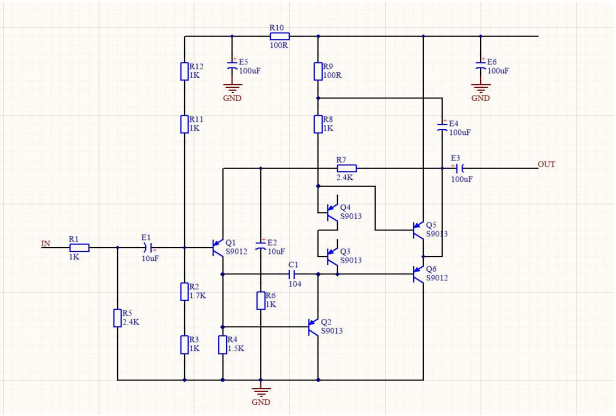

图 2.1.3 - 晶体管放大电路

电路中采用的是 PNP 三极管进行作为放大管，输入正弦波与三极管之间使用阻容耦合，使用一个 10uf 的铝电解电容连接，根据电容隔直通交的特性，它可以滤除掉输入中的直流分量，一定程度上减小了干扰。再看电阻 R7 和 R8，这两个电阻是用来调节放大器的静态工作点的电阻。

放大后从 Q1 的集电极输出，与第二级放大器直接耦合，再次放大一次，三极管的基极和

集电极间并联一个 330pf 的瓷片电容用于防止自激振荡。经过两级放大后到互补输出级，为了消除交越失真，原本想要采用两个二极管来增大死区时间，但是发现二极管 1N4148 的导通电压为 0.6V，与三极管 9013 不同，所以选用两个 9013 的 be 极组成， C1 采用一个 100uf 的铝电解电容作为自举电容抬升电压，这样就省去了一个 Vcc 供电。输出端接一个 100uf 的铝电解电容用于滤波。

2.1.2 波形的切换控制

我们的硬件电路用了 4 个模拟开关通道控制电路的通断，对应单片机中我们也引出 4 个 IO 接口，由于模拟开关 CD4066 驱动电压为 5v，我们在单片机的 IO 口和模拟开关控制接口之间加了一级缓冲器，进行电平转换。

2.1.3 电压采集以 THD 的计算

首先是电压的输出采集，由于晶体管放大电路的输出电压范围约为-2v－+2v，而单片机的 ADC 采样电压范围是 0v-3.3v，因此，需要将晶体管放大电路的输出电压经过抬升、衰减(或放大)电路后再接入单片机的 AD 采样接口。

接下来是 THD 的计算，根据 THD 的计算公式:

$$\mathrm{THD}=\frac{\sqrt{U_{o2}^2+U_{o3}^2+U_{o4}^2+\mathrm{L}+U_{on}^2}}{U_{o1}}\times 100\%$$

公式中的分母含有各次谐波分量，但实际采集的过程中，很难将所有谐波都采集到，所以我们只采集前 5 次谐波，后面的谐波分量幅值太小，可以忽略不计。

由于 5 次谐波的频率为 5kHz，根据采样定理，实际应用中至少要保证采样频率为 20kHz，在综合考虑 THD 的计算精度、谐波频率、单片机的时钟频率、以及 DSP 库能够计算采样宽度等因素，最终我们选择的采样频率为 32768Khz，采样宽度为 4096，分辨率可以达到 8Hz/采样宽度。

2.2 参数整定计算

发射极-集电极静态电压的计算:

$$U_{CEQ}=V_{cc}-\frac{V_{cc}-0.7-U_{1BQ}}{R_4}$$

R7、R8 是调整静态工作点的电阻，使用 Multisim 仿真，将预估整定的 R7、R8 的阻值带入仿真电路中，再通过观察仿真的现象对参数进行调整。经过计算和仿真，我们得出的失真参数如下：

顶部失真：R7 47K，R8：27K；底部失真：R7：100K R8:10K 。

第二级放大电路中用 R4、R5 来调整放大增益，R4 为 2.4K，R5 位 10 欧姆，放大倍数为 240 倍，经过放大后理论电压为 4.8V，由于供电电压为 5V，所以一定会发生失真现象，为了消除这个双向失真，在信号接入口那边接入一个信号衰减电路，从而使它能够输出不失真的波形，而直接短接掉电阻就可以输出双向失真的波形。

推挽输出级那边用二极管来消除交越失真，由于 1N4148 导通电压时 0.6V，而两个二极管需要 0.7v

的导通电压，所以将两个二极管直接用用三极管替换，将这两个三极管短接就可以产生交越失真。

2.3 核心电路设计仿真

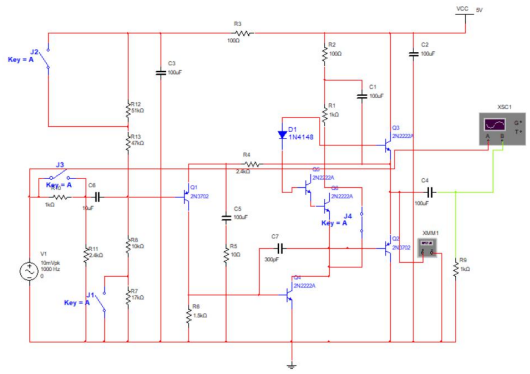

图 2.3.1-晶体管放大电路仿真电路图

2.4 电路实现调试测试

给电路板接上电源，输入一个 1kHz，峰峰值为 20mV 的正弦波，输出接示波器，用开关选择电阻，观察各种阻值对的波形是否满足要求。

三、系统软件设计分析

3.1 系统总体工作流程

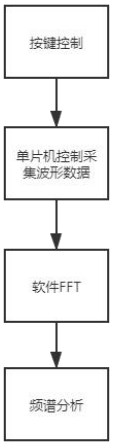

图 3.1.1-按键软件控制流程图

3.2 主要模块程序设计

3.2.1 波形切换控制模块

我们选用 STM32 进行波形的切换控制，通过 GPIO 输出逻辑电平来控制模拟开关的通断。进而产生不同的波形。

3.2.2 软件 FFT

我们选用 STM32 对输出信号进行 AD 采样，得到采样序列，然后将采样序列输入到 STM32 的 DSP 库进行快速傅里叶变换，得到信号的频谱，从频谱中挑选出基波以及各次谐波分量，进而计算出 THD。

四、竞赛工作环境条件

我们使 STM32 作为主控芯片，用于 FFT 和 UI 控制，借助 MDK 集成开发环境进行编程。在硬件方面，使用焊台焊接万能电路板，完成电路图的连接，使用信号发生器输入 1kHz，20mV 的正弦信号。使用数字示波器测试结果。

五、作品成效总结分析

5.1 系统测试性能指标

能够生成无失真放大后的正弦波、顶部、底部、双向、交越波形，波形平滑。波形如下图 5.4.1-5.4.5 所示。性能指标见图 5.1.6。经 FFT 后能够较好的完成从时域到频域的变换，总谐波失真测试计算较准。

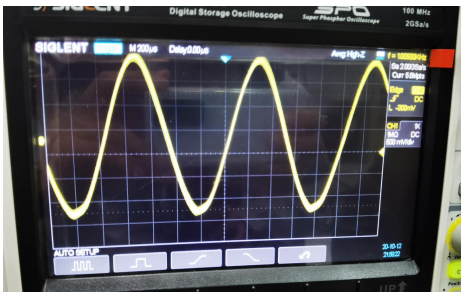

图 5.1.1 - 标准正弦波

图 5.1.2 - 底部失真的正弦波

图 2.1.3 - 顶部失真正弦波

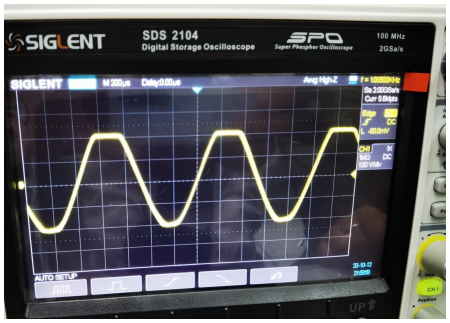

图 5.1.4 - 双向失真的正弦波

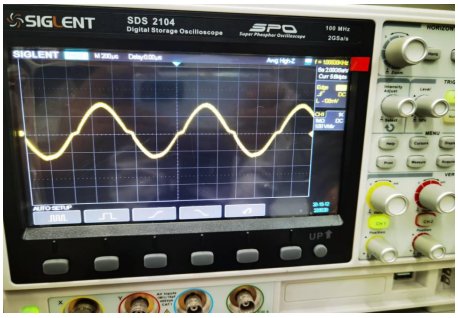

图 5.1.5 - 交越失真正弦波

波形	正弦波	顶部失真	底部失真	双向失真	交越失真	三角波	方波
Vpp/V	3.28	3.28	3.5	4.01	3.02	3.0	3.0
F/KHz	1.00681	1.00682	1.00682	1.00682	1.00682	1.0	1.0
THD	0.000%	13.236%	5.402%	12.527%	5.240%	13.832%	48.300%

图 5.1.6-晶体管放大电路的性能指标

注：图 5.1.6 所示的晶体管放大电路的性能指标，方波和三角波由函数信号发生器产生，仅为展示 THD 测量的准确度。其余波形由晶体管放大电路产生。

5.2 成效得失对比分析

我们所选取的方案是经过我们商讨后的在短时间内做到最好的方案，这个方案舍弃了一些精度，但很大程度上提高了我们的调试速度。

5.3 创新特色总结展望

经过四天三夜的奋战，我们终于完成了这次电子设计竞赛的 E 赛题，这个过程是艰难而劳累的，队友们都很辛苦，但是我们的辛苦是值得的，在这次比赛中，我们对模拟电路的理解更加深刻，增强了专业基础和实践操作的能力，还培养了团队合作的能力。我们这次设计出一个完整的集成运放电路，将硬件和软件的各个模块很好的结合到了一起。赛后有时间，我们会将之前想到的较为精准的方案具体实现一遍。

六、参考资料

[1] 童诗白,华程英.模拟电子技术基础（第四版）[M].北京:高等教育出版社,2009。

[2] 阎石.数字电子技术基础（第五版）[M].北京:高等教育出版社,2009。

[3] 黄智伟,王彦,陈文光等.全国大学生电子设计竞赛训练教程[M].北京:电子工业出 版社,2007。

[4] 高吉祥,唐朝京.全国大学生电子设计竞赛培训系列教程（电子仪器仪表设计）[M]. 北京:电子工业出版社,2007。

[5] 郭天祥.新概念 51 单片机 C 语言教程.入门、提高、开发[M].北京：电子工业出版社，2009。

[6] 梁明理.电子线路（第五版）[M].北京：高等教育出版社 2008。

[7] 刘火良、杨森.STM32 库开发实战指南 [M].北京：机械工业出版社 2013.6。

[8] 吴镇扬.数字信号处理（第三版）[M].北京：高等教育出版社 2016.7。

附录：

关键部分源程序代码：

```
#define SW1_GPIO      GPIOD//GPIOB
#define SW1_GPIO_PIN  GPIO_PIN_3//GPIO_PIN_5
#define SW2_GPIO      GPIOD//GPIOB
#define SW2_GPIO_PIN  GPIO_PIN_2//GPIO_PIN_6//GPIO_PIN_3
#define SW3_GPIO      GPIOD//GPIOB
#define SW3_GPIO_PIN  GPIO_PIN_4//GPIO_PIN_7//GPIO_PIN_4
#define SW4_GPIO      GPIOD//GPIOB
#define SW4_GPIO_PIN  GPIO_PIN_5//GPIO_PIN_4//GPIO_PIN_6

#define ON 1
#define OFF 0
#define Switch1(state) ((state) == ON?HAL_GPIO_WritePin(SW1_GPIO,SW1_GPIO_PIN,GPIO_PIN_SET):HAL_GPIO_WritePin(SW1_GPIO,SW1_
#define Switch2(state) ((state) == ON?HAL_GPIO_WritePin(SW2_GPIO,SW2_GPIO_PIN,GPIO_PIN_SET):HAL_GPIO_WritePin(SW2_GPIO,SW2_
#define Switch3(state) ((state) == ON?HAL_GPIO_WritePin(SW3_GPIO,SW3_GPIO_PIN,GPIO_PIN_SET):HAL_GPIO_WritePin(SW3_GPIO,SW3_
#define Switch4(state) ((state) == ON?HAL_GPIO_WritePin(SW4_GPIO,SW4_GPIO_PIN,GPIO_PIN_SET):HAL_GPIO_WritePin(SW4_GPIO,SW4_
```

波形控制 1/5

```
void DisplayNormalWave()
{
    Switch1(ON);
    Switch2(OFF);
    Switch3(OFF);
    Switch4(OFF);
}
```

波形控制 2/5

```
void DisplayWave()
{
    float THD = 0.0;

    DisplayNormalWave();    //正常
    OLED_Clear();

    OLED_ShowString(0,0,(uint8_t *)"Normal Wave");

    CalculateTHDAve(&THD,NPT);

    ClearStrBuf();
    sprintf((char *)str_buff,"%7.3f %%",THD*100);//将数据格式化为字符串

    OLED_ShowString(46,4,str_buff);
    HAL_Delay(4500);
    BEEP_ON();
    HAL_Delay(500);
    BEEP_OFF();


    DisplayTopDistortionWave(); //顶部
    OLED_Clear();
    OLED_ShowString(0,0,(uint8_t *)"Top Distortion Wave");

    CalculateTHDAve(&THD,NPT);

    ClearStrBuf();
    sprintf((char *)str_buff,"%.3f %%",THD*100);//将数据格式化为字符串   OLED_ShowString(46,4,str_buff);
```

波形控制 3/5

```
    OLED_ShowString(46,4,str_buff);

    HAL_Delay(4500);
    BEEP_ON();
    HAL_Delay(500);
    BEEP_OFF();

    OLED_Clear();
    DisplayButtonDistortionWave(); //底部
    OLED_ShowString(0,0,(uint8_t *)"Button Distortion Wave");

    CalculateTHDAve(&THD,NPT);
    ClearStrBuf();
    sprintf((char *)str_buff,"%.3f %%",THD*100);//将数据格式化为字符串

    OLED_ShowString(46,4,str_buff);
    HAL_Delay(4500);
    BEEP_ON();
    HAL_Delay(500);
    BEEP_OFF();


    OLED_Clear();
    OLED_ShowString(0,0,(uint8_t *)"Two Way Distortion Wave");
    DisplayTwoWayDistortionWave();
    CalculateTHDAve(&THD,NPT);

    ClearStrBuf();
```

波形控制 4/5

```
    sprintf((char *)str_buff,"%.3f %%",THD*100);//将数据格式化为字符串
    OLED_ShowString(46,4,str_buff);
    HAL_Delay(4500);
    BEEP_ON();
    HAL_Delay(500);
    BEEP_OFF();

    DisplayCrossoverDistortionWave(); //交越
    OLED_Clear();
    OLED_ShowString(0,0,(uint8_t *)"Crossover Distortion Wave");

    CalculateTHDAve(&THD,NPT);

    ClearStrBuf();
    sprintf((char *)str_buff,"%.3f %%",THD*100);//将数据格式化为字符串

    OLED_ShowString(46,4,str_buff);
    HAL_Delay(4500);
    BEEP_ON();
    HAL_Delay(500);
    BEEP_OFF();
}
```

波形控制 5/5

```
void ADC_DMAStart()
{
  HAL_Delay(500);

  HAL_ADC_Start_DMA(&hadc1,(uint32_t *)adc_buff,NPT);

  HAL_Delay(500);
  HAL_ADC_Stop_DMA(&hadc1);




  for(int i=0;i<NPT;i++)
  {
      fadc_buf[i] = (adc_buff[i] & 0x0fff) * 1.0;
  }
}
```

电压采集 1/1

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

#define NPT 4096
#define PI2 6.28318530717959
#define Fs  (32768)
#define R   Fs/(NPT*1.0)
#define ifftFlag 0
#define doBitReverse 1

typedef struct{
  uint16_t num;
  float   value;
  float   hz;
}disBuf;

float fft_hz[5];
float fft_Value[5];

disBuf disFFT[(NPT>>1)];   //频谱线数据

uint16_t adc_buff[NPT]={0};
float fadc_buf[NPT] = {0.0};
float FFT_InputBuf[NPT*2];
float FFT_OutputBuf[NPT];
```

THD 计算 1/5

```
void FFT_Shift(float *SampleValueBuf,float *OutPutBuf,uint32_t len)
{
  for(int i = 0;i<len;i++)
  {
    FFT_InputBuf[2*i] = (float)SampleValueBuf[i];//* 3.3/4096.0;
    FFT_InputBuf[2*i + 1] = 0.0;
  }

  arm_cfft_f32(&arm_cfft_sR_f32_len4096,FFT_InputBuf,ifftFlag,doBitReverse);

  arm_cmplx_mag_f32(FFT_InputBuf, OutPutBuf,len);
}
```

THD 计算 2/5

```
240 void CalculateTHD(float *fft_outputbuf,float *THD,float len)
241 {
242     int FFT = 0;
243     float a = 0.0;
244 
245     for(int i = 0;i<5;i++)
246     {
247        fft_hz[i] = 0.0;
248        fft_Value[i] = 0.0;
249     }
250     for(i=1;i<len-1;i++)
251     {
252       if((fft_outputbuf[i]>fft_outputbuf[i-1]&&fft_outputbuf[i]>fft_outputbuf[i+1]))
253       {
254         if(fft_outputbuf[i]>(float)62060.6)
255         {
256           disFFT[FFT].num =i;
257           disFFT[FFT++].value =fft_outputbuf[i];
258         }
259       }
260     }
261     for(i=0;i<FFT/2;i++)
262     {
263       if((float)(disFFT[i].num-1)*(Fs*1.0)/(len*1.0)>=5200.0f)
```

THD 计算 3/5

```
264       {
265           break;
266       }
267       else
268       {
269           fft_hz[i]=(float)((disFFT[i].num)*Fs/len);
270           fft_Value[i]=(disFFT[i].value + disFFT[FFT-1-i].value)/len*2;
271       }
272 
273     }
274 
275     arm_sqrt_f32(fft_Value[1]*fft_Value[1]+fft_Value[2]*fft_Value[2]+fft_Value[3]*fft_Value[3
276     if(fabs(fft_Value[0])<1e-6)
277       *THD = 0.0;
278     else
279       *THD=a/fft_Value[0];
280 
281     if(*THD >= 0.483f)
282       *THD = 0.483f;
283 
284     printf1("\r\n%.3f\r\n",*THD);
285 }
```

THD 计算 4/5

```
287 float CalculateTHDAve(float *THD,float len)
288 {
289   float _THD[5] = {0.0,0.0,0.0,0.0,0.0};
290     *THD = 0;
291   for(int i = 0;i<5;i++)
292   {
293     ADC_DMAStart();
294 
295     FFT_Shift(fadc_buf,FFT_OutputBuf,NPT);
296     CalculateTHD(FFT_OutputBuf,&_THD[i],NPT);
297       *THD += _THD[i];
298   }
299   *THD /= (5.0f);
300 
301   return *THD;
302 }
```

THD 计算 5/5