**2021年（第十届）全国大学生电子设计竞赛报告**

**A题**  **信号失真度测量装置**



**2021年11月4日**

# 信号失真度测量装置

摘要

本设计以MSP432E401Y单片机为核心，通过对输入信号进行同相放大、等效采样、FFT变换和频谱搬移等操作，精确测算出输入信号的非线性失真程度，并可在本地屏幕以及手机上快速显示出信号失真度、一个周期的信号波形以及归一化幅值。经测试，本设计功能齐全，测量耗时极短，并利用等效采样的方法能够对400kHz信号进行谐波分析，超过拓展要求的四倍，而且THD的测量与理论值始终保存1%以内的误差，一次完整的信号分析与显示总共耗时不足一秒。在工程上用等效采样来分析谐波的方式比等效采样节约硬件成本，值得推广。

**目 录**

[信号失真度测量装置 1](#_Toc90471451)

[1.1 设计任务与要求 1](#_Toc90471452)

[1.1.1设计任务 1](#_Toc90471453)

[1.1.2设计要求 1](#_Toc90471454)

[1.2 方案设计与论证 2](#_Toc90471455)

[1.2.1信号采样 2](#_Toc90471456)

[1.2.2 FFT数据类型对比 2](#_Toc90471457)

[1.3 理论分析与计算 3](#_Toc90471458)

[1.3.1 信号处理 3](#_Toc90471459)

[1.3.1.1 信号放大电路 3](#_Toc90471460)

[1.3.1.2 加法器电路 3](#_Toc90471461)

[1.3.2 等效采样 3](#_Toc90471462)

[1.3.3 FFT相关概念 4](#_Toc90471463)

[1.3.3.1 窗函数 4](#_Toc90471464)

[1.3.3.2 频谱分辨率 5](#_Toc90471465)

[1.3.4输入信号失真度分析与计算 5](#_Toc90471466)

[1.3.4.1 信号的失真程度 5](#_Toc90471467)

[1.3.4.2 归一化幅值 5](#_Toc90471468)

[1.4 系统软、硬件设计 6](#_Toc90471469)

[1.4.1 系统硬件设计 6](#_Toc90471470)

[1.4.1.1 电源 6](#_Toc90471471)

[1.4.1.2 信号处理电路 6](#_Toc90471472)

[1.4.1.3 主控MCU 6](#_Toc90471473)

[1.4.1.4 无线传输模块 6](#_Toc90471474)

[1.4.2 系统软件设计 6](#_Toc90471475)

[1.4.2.1 MSP432主程序 6](#_Toc90471476)

[1.4.2.2 ESP32主程序 7](#_Toc90471477)

[1.4.2.3 APP主程序 7](#_Toc90471478)

[1.5 系统测试及结果 8](#_Toc90471479)

[1.5.1 测试仪器 8](#_Toc90471480)

[1.5.2 测试方法 8](#_Toc90471481)

[1.5.3 性能指标 8](#_Toc90471482)

[1.6 总结与展望 8](#_Toc90471483)

[附 件 9](#_Toc90471484)

[附件1 ESP32工作流程图 9](#_Toc90471485)

[附件2 APP工作流程图 10](#_Toc90471486)

[附件3 电源电路原理图 11](#_Toc90471487)

[附件4 程控放大器电路原理图 11](#_Toc90471488)

[附件5 加法器电路原理图 12](#_Toc90471489)

[附件6 MSP432部分代码 12](#_Toc90471490)

[附件7 ESP32部分代码 22](#_Toc90471491)

[附件8 APP 部分代码 23](#_Toc90471492)

## 1.1 设计任务与要求

### 1.1.1设计任务

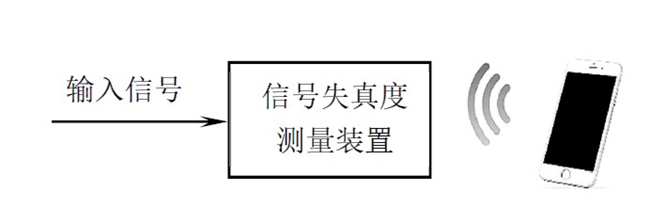
设计制作信号失真度测量装置，对来自函数/任意波形发生器的周期信号（以下简称为输入信号）进行采集分析，测得输入信号的总谐波失真THD（以下简称为失真度），并可在手机上显示测量信息。测量装置系统组成示意图如图1 所示。

图1-1 测量装置系统组成示意图

### 1.1.2设计要求

1. 基本要求：

1.输入信号的峰峰值电压范围：300mV~600mV

2.输入信号基频：1kHz

3.输入信号失真度范围：5% ~ 50%

4.要求对输入信号失真度测量误差绝对值，Δ=|THDX -THDO|≤5%

5.显示失真度测量值THDX

6.失真度测量与显示用时不超过10 秒

1. 发挥部分：

1.输入信号的峰峰值电压范围：30mV ~ 600mV

2.输入信号基频范围：1kHz ~100kHz

3.测量并显示输入信号失真度THDX值，要求Δ=|THDX -THDO|≤3%

4.测量并显示输入信号的一个周期波形

5.显示输入信号基波与谐波的归一化幅值，只显示到5 次谐波

6.在手机上显示输入信号THDX 值、一个周期波形、基波与谐波的归一化幅值

7.其他

## 1.2 方案设计与论证

本系统首先对输入信号进行程控放大，之后使用反相放大器和加法器将信号调整到适当的范围，最后使用MSP432E401Y板载12位高精度ADC对信号进行采集。系统整体框架如图所示：

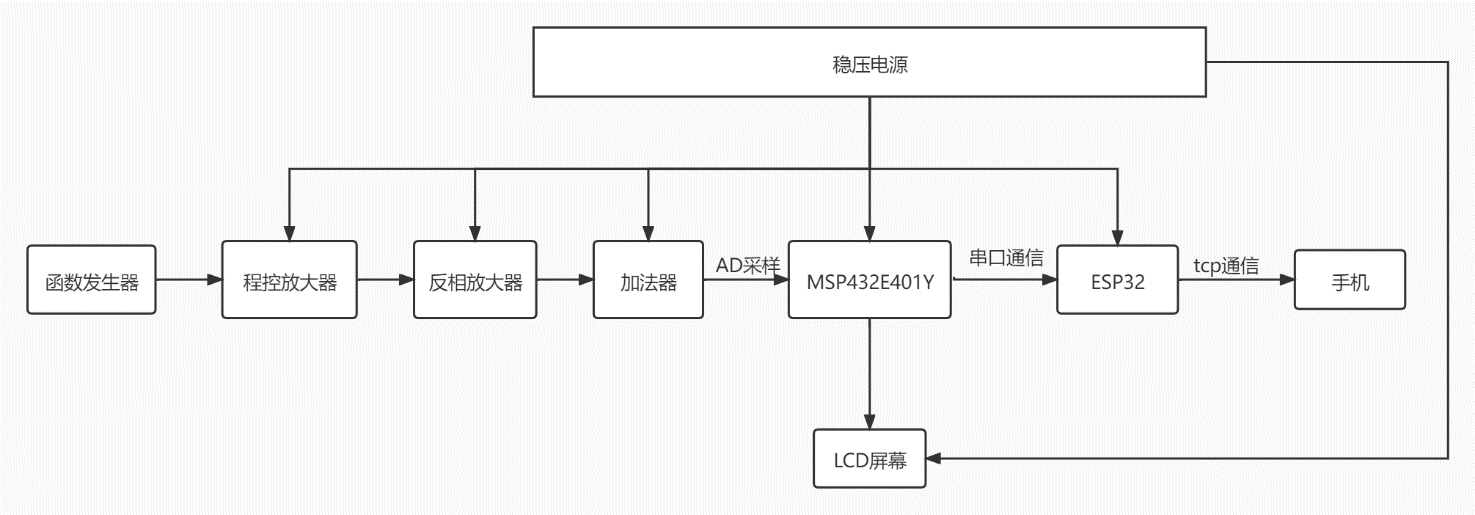


图1-2 信号失真度测量装置系统框图

### 1.2.1信号采样

方案一：直接采样。

以1M SPS固定采样率进行采样，对采集数据直接进行FFT变换计算基波和各谐波分量，最后计算出信号的失真度。优点：操作简单；缺点：板载ADC性能有限，频谱分辨率较低，误差较大。

方案二：等效采样。

根据输入信号基波频率变换采样率，同时使用等效采样算法对采集到的数据进行重组进而恢复出原始信号，最后对恢复出的信号进行FFT变换计算基波和各谐波分量，进而得到输入信号的失真度。优点：能够有效降低对ADC采样率和数据存储速度的要求；缺点：采样时间有所增加，测量时间变长，软件开发难度大，对算法的要求高。

综合考虑对测量精度和速度的要求，采用方案二。

### 1.2.2 FFT数据类型对比

信号采集方案一和方案二最终均需要对采集到的数据进行FFT运算，MSP432可使用ARM提供的DSP库，在该DSP库中FFT运算数据类型可选择定点数或浮点数。

方案一： 16位定点数。

输入数据和FFT运算结果均使用16位整形存储。优点：对存储空间的需求较低，计算量小。缺点：运算结果精度低，只能进行整数的运算。

方案二： 32位浮点数。

输入数据和FFT运算结果均使用32位浮点数存储。优点：运算结果精度高，可以进行小数之间的计算。缺点：对存储空间需求高，计算量大。

综合考虑计算精度和存储空间，采用方案二。

## 1.3 理论分析与计算

### 1.3.1 信号处理

1.3.1.1 信号放大电路

函数发生器输出信号的峰峰值范围为30mVpp – 600mVpp，而ADC在采集1V – 2V的 信号时较为精确，因此需要对输入信号进行一定程度的放大，信号放大方案如下表所示：

表1-1 信号放大方案

|  |  |  |
| --- | --- | --- |
| **输入信号** | **总放大倍数** | **输出信号** |
| 30mVpp – 60mVpp | 40 | 1.2Vpp – 2.4Vpp |
| 60mVpp – 120mVpp | 20 | 1.2Vpp – 2.4Vpp |
| 120mVpp – 300mVpp | 8 | 0.96Vpp – 2.4Vpp |
| 300mVpp – 600mVpp | 4 | 1.2Vpp – 2.4Vpp |

1.3.1.2 加法器电路

由信号放大电路输出的信号存在负电压，而MSP432E401Y板载ADC无法采集负电压，因此需要使用加法器为其添加一个1.4V的直流偏移。

### 1.3.2 等效采样

等效采样是把周期性或准周期性的高频、快速信号变换为低频的慢速信号的技术，该技术减少了单位时间内的采样点数，降低了对实际采样速率、数据存储速度、信号处理速度的要求。在电路上只对取样前的电路具有高频的要求，简化了整个系统的设计难度。

下面举例介绍等效采样技术：若采用实时采样技术对下图信号的连续两个周期进行采样，共40个采样点，40个采样点的分布如图3所示。

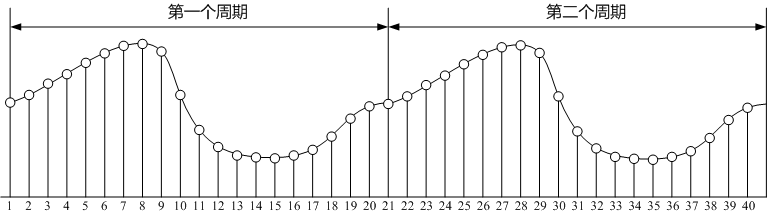


图1-3 两个信号周期连续实时采样获取的采样点

如图4所示，我们也可以只提取红色采样点：第一个周期中取序号为1的采样点，第 二个周期取序号2的采样点，第三个周期取序号3的采样点，以此类推，直到经过40个周期采集完40个点，再按照图5的顺序排列，进而拼凑出一个完整的波形，再使用数字信号处理的数据处理方法对信号进行分析。

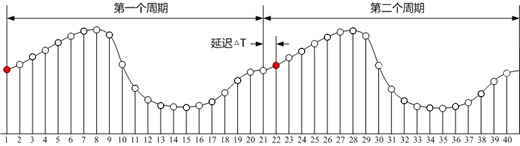


图1-4 周期信号等效采样原理

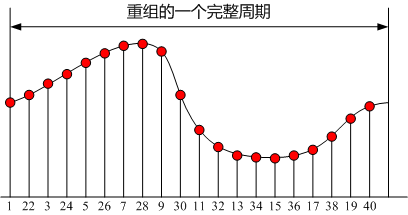


图1-5 采用等效采样技术重组的一个完整信号周期

### 1.3.3 FFT相关概念

1.3.3.1 窗函数

采样过程中不可避免地会产生频谱泄露，通过加窗可以很好地缓解该问题，实际应用中经常使用矩形窗和汉宁窗。矩形窗的主瓣较宽，带内较为平坦，可以较为准确地得到相应频率点的幅度值；汉宁窗的主瓣较窄，旁瓣滚降率较大，频谱分辨率更高。本系统需要确定输入信号的基频频率，因此需要有较高的频谱分辨率，而要精确地计算失真度就需要使各频率分量的幅度值尽可能准确。可见，单一地使用矩形窗或汉宁窗均无法满足实际需要。因此，本系统采用双窗法来兼顾频谱分辨率和幅度准确度，即使用汉宁窗来确定信号基频频率，之后使用矩形窗计算相应频率点的幅度。

1.3.3.2 频谱分辨率

DFT频域相邻刻度之间的实际频率之差即为频谱分辨率，若用表示，则：

假设采样率为1024Hz，采样时间为1s，则DFT频谱分辨率可达到1024Hz/1024 = 1Hz；如果采样时间增加到2秒，则DFT频谱分辨率可达到0.5Hz。在采样率不变的情况下，通过延长采样时间可以增加采样点数，进而提高频谱分辨率。

### 1.3.4输入信号失真度分析与计算

1.3.4.1 信号的失真程度

当放大器输入为正弦信号时，放大器的非线性失真表现为输出信号中出现谐波分量，即出现谐波失真，通常用“总谐波失真THD”定量分析放大器的非线性失真程度。

若放大器的输入交流电压为ui = Ui cos𝝎t ，出现谐波失真的放大器输出交流电压为

，则uo的总谐波失真度定义为：

1.3.4.2 归一化幅值

当输入信号的基波幅值为，各次谐波幅值分别为、、…，基波与谐波的归一化幅值为：1、、、……。

## 1.4 系统软、硬件设计

### 1.4.1 系统硬件设计

1.4.1.1 电源

本设计采用LM317和LM337设计可调稳压电源，由于LM337和LM317自身具有输出短路保护、过热关断等功能，且PCB版图中做了大量的铺铜散热处理，使得该电路可在高负载(2A)下长时间(>2h)稳定工作。相关原理图见附件3。

1.4.1.2 信号处理电路

信号处理电路中的程控放大器使用两片INA821级联而成，放大倍率分别为2和5，通过两片继电器芯片控制输入信号的通路，进而控制总放大倍率。由于INA821的压摆率较低（2V/us），无法直接将输入信号放大至1V-2V的范围，因此我们在程控放大器后增加一级固定增益的反相放大器来弥补其不足，固定增益的反相放大器由 OPA140搭建，增益为4，该增益下带宽约为2Mhz。相关原理图见附件4。

加法器电路使用电容和电阻网络构成，原理如附件5所示。由于其输入端等效为一个高通滤波器，而输入信号范围为1kHz–400kHz，因此需要保证其截止频率低于1kHz。本设计选取C=1uF，R1=1.9K，R2=1K，等效高通滤波器截止频率为242Hz < 1kHz。

1.4.1.3 主控MCU

本方案选择MSP432E401Y单片机作为系统的主控，其板载两个12 位 SAR、2M SPS采样率的ADC模块，能够精确地测量1kHz - 100kHz的信号。

1.4.1.4 无线传输模块

无线传输部分由ESP32完成，ESP32是一款集成 WiFi 功能的微控制器，支持 802.11 b/g/n/d/e/i/k/r 等WIFI协议。MSP432E401Y通过串口将数据打包发给ESP32模块，ESP32将数据通过TCP连接将数据发送到手机，实现和手机之间的无线通信。

### 1.4.2 系统软件设计

1.4.2.1 MSP432主程序

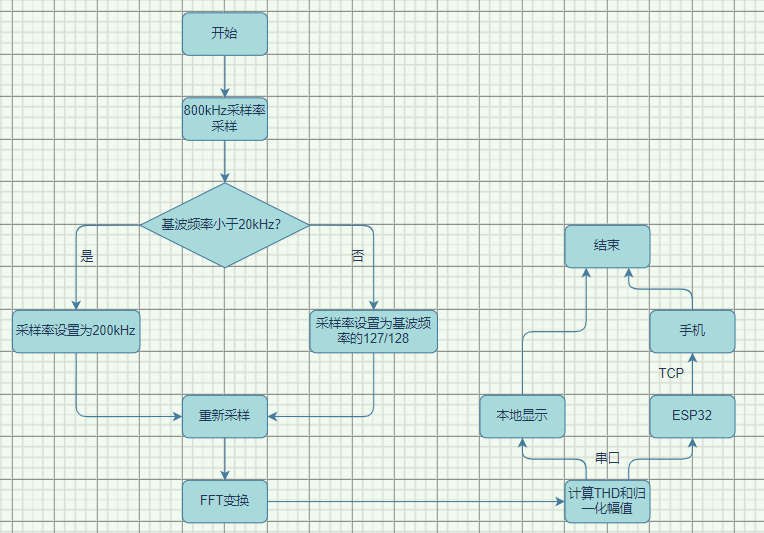


图1-6 主程序流程图

系统初始状态先设定为800kHz采样率进行采样，大致判断输入信号的频率范围。对于信号的处理分为两种情况：当信号基波频率小于20kHz时，我们将采样率调整为200kHz并使用实时采样算法，之后进行FFT变换，精确测算基波频率、各谐波归一化幅值，最后计算总谐波失真度；当信号的基波频率大于20kHz时，我们将采样率调整为输入信号基波频率的并使用等效采样算法，之后进行FFT变换，根据频谱关系计算出等效采样率，计算公式为如下：

根据此公式，当基波频率为400kHz时，等效采样率可以达到50MHz。

为了消除偶然误差，系统在一次采样开启后进行10次采样，去除与10次THD均值相差最大的两个数据后对剩余8个数据重新取均值，并将结果作为最终的测量值。在获得THD值和各谐波分量以及基波频率后，MSP432E401Y单片机通过串口将数据发送给屏幕和ESP32，ESP32将数据再发送给手机，完成一次完整的信号失真度测量与分析。

1.4.2.2 ESP32主程序

ESP32模块负责从串口接收MSP432发送的数据包，并将数据包通过TCP连接发送给手机，其工作 流程如附件1所示。

1.4.2.3 APP主程序

手机APP负责接收ESP32模块发送的数据包，并解析其具体内容，最终将THD、输入信号波形和归一化幅值显示出来，其工作流程如附件2所示。

## 1.5 系统测试及结果

### 1.5.1 测试仪器

表1-2 测试仪器

|  |  |
| --- | --- |
| 仪器名称 | 型号规格 |
| RIGOL函数信号发生器 | 60MHz 500Msa/s |

### 1.5.2 测试方法

用自制的电源为本系统所有设备供电，通过调节函数发生器的谐波输出功能来输入信 号，在本地显示器和手机上读出测得的非线性失真程度数据并记录。

### 1.5.3 性能指标

表1-3 THD测量精度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| THD标称值/% | 10.0 | 20.0 | 30.0 | 40.0 |
| THD测量值/% | 10.2469 | 20.1043 | 30.4935 | 40.3354 |
| Δ/% | 0.2469 | 0.1043 | 0.4935 | 0.3354 |

## 1.6 总结与展望

本系统以MSP432E401Y单片机为核心，由其片内ADC采集数据，经计算得出输入信号的非线性失真程度。在系统设计中，力求简化硬件电路设计，充分发挥软件编程灵活的特点来满足系统设计的要求。

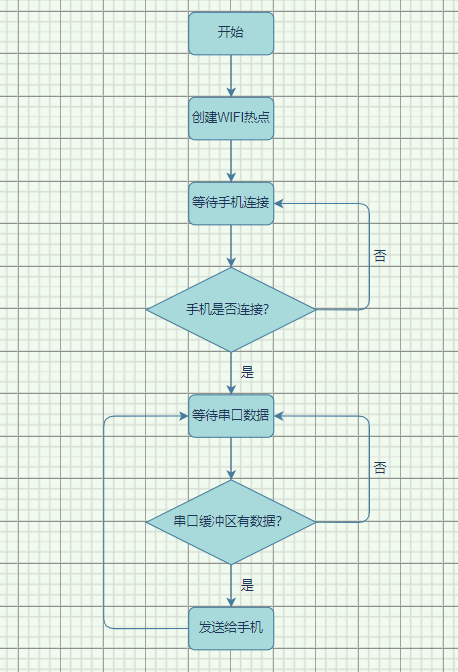
从测试结果看，本设计完成并超过了题目的要求，信号失真度测量精度高、误差小、速度快，另外增加屏幕显示使得数据清晰明了。

在本设计中我们创新性地使用了等效采样算法，在低采样率的情况下成功采集并重组出原始高频信号，其在工业产品的设计中存在实际应用价值。

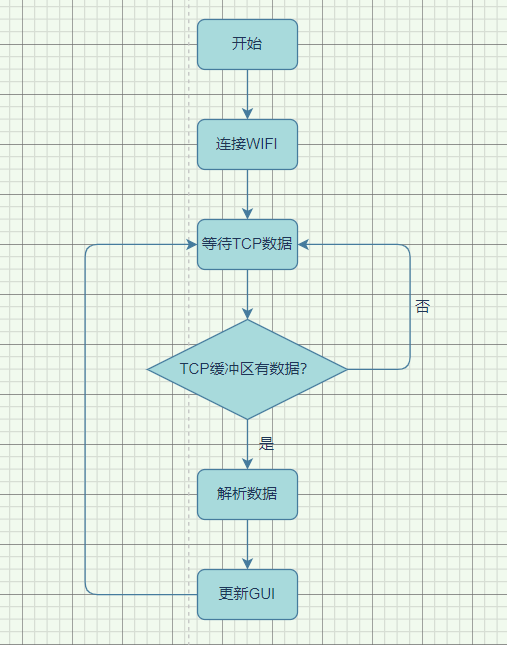
## 

# 附 件

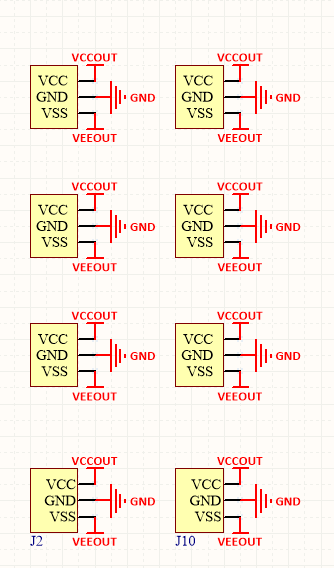
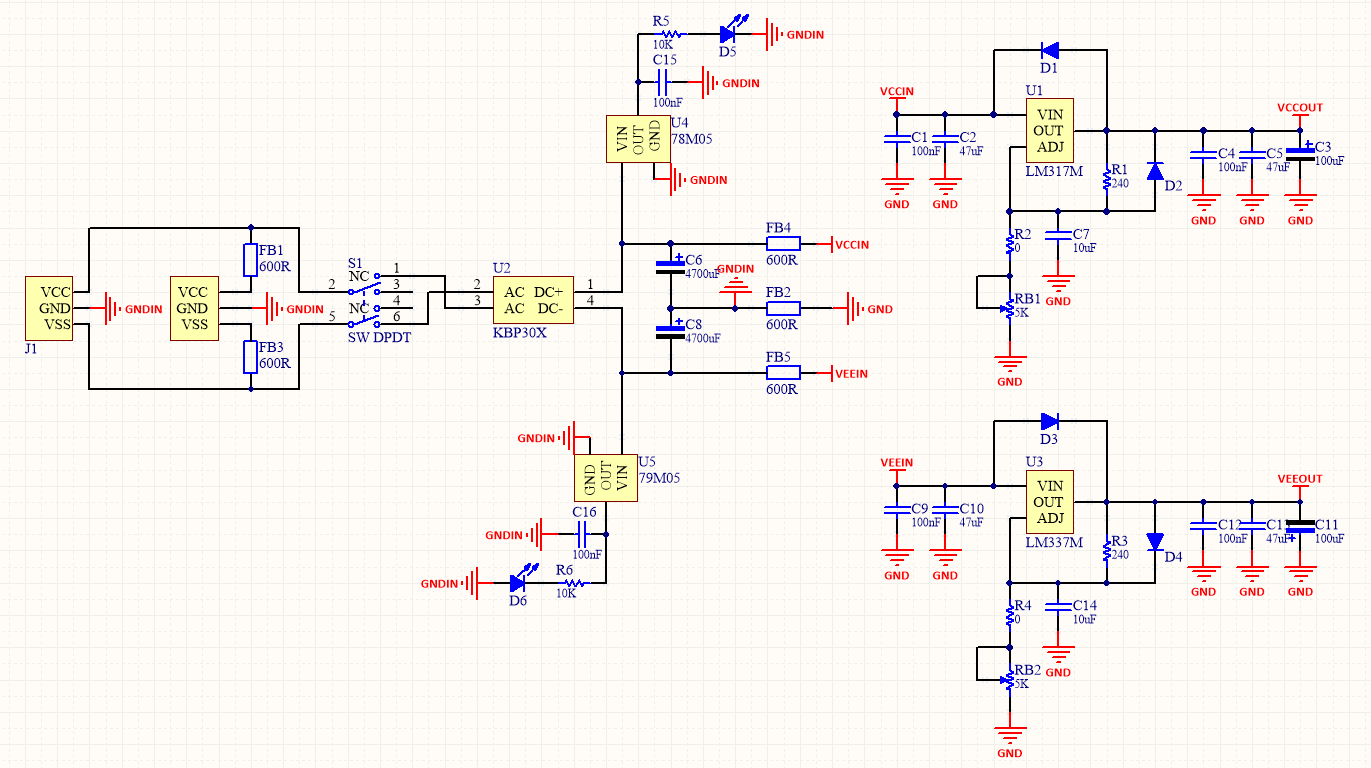
## 附件1 ESP32工作流程图



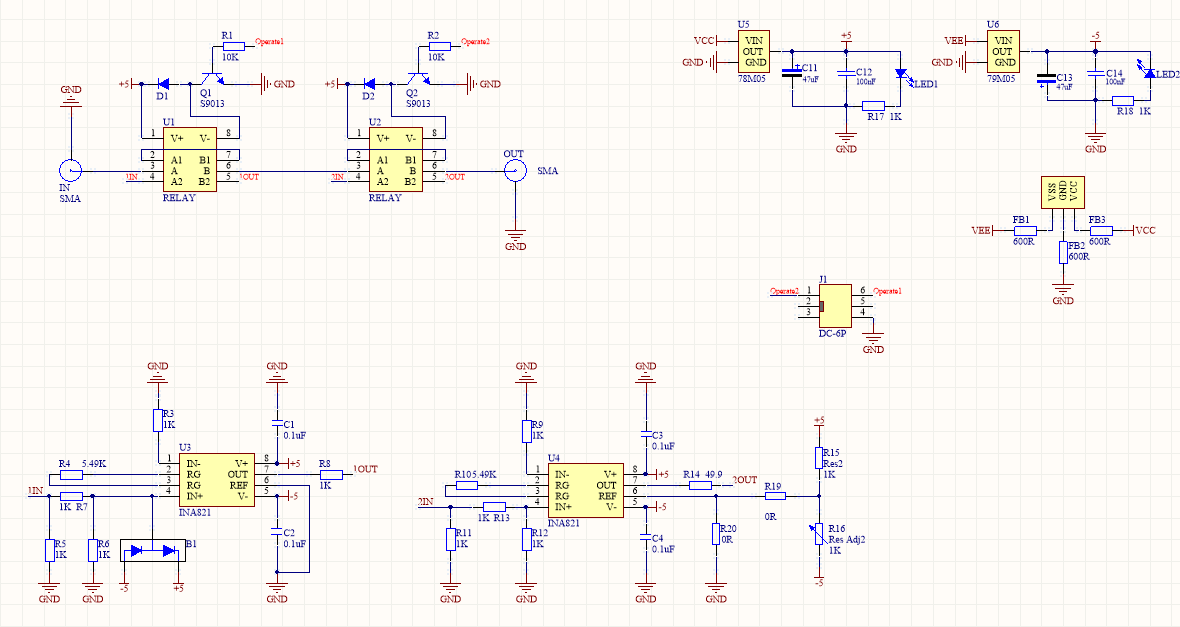
## 附件2 APP工作流程图



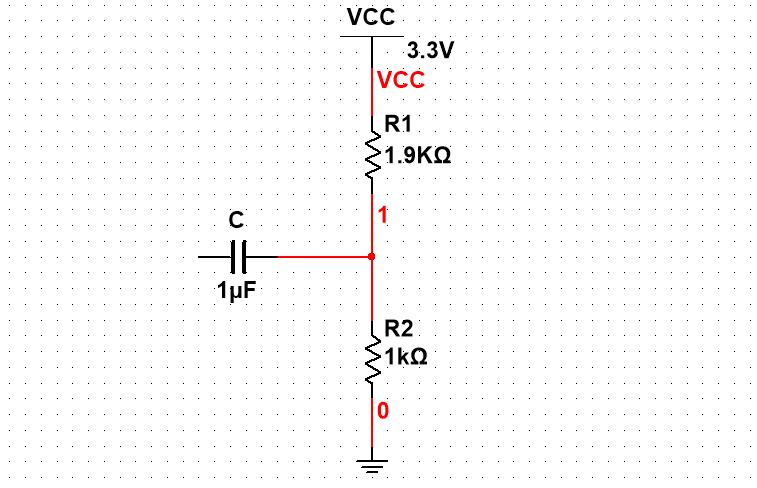
## 附件3 电源电路原理图



## 附件4 程控放大器电路原理图



## 附件5 加法器电路原理图



## 附件6 MSP432部分代码

（1）头文件及变量表

1. //变量表与引用头文件
2. #include <ti/devices/msp432e4/driverlib/driverlib.h>
3. #include <stdio.h>
4. #include <stdint.h>
5. #include <stdlib.h>
6. #include <stdbool.h>
7. #include <math.h>
8. #include "arm\_math.h"
9. #include "uartstdio.h"
10. #include "arm\_const\_structs.h"
11. /\* Define for Samples to be captured and Sampling Frequency \*/
12. //宏定义表
13. #define NUM\_SAMPLES 1024
14. #define My\_PI 3.1415926f
15. #define Cir\_Times 1
16. #define IFFTFLAG 0
17. #define BITREVERSE 1
18. //DMA-FFT相关数据
19. static uint32\_t dstBufferA[NUM\_SAMPLES];
20. static uint32\_t dstBufferB[NUM\_SAMPLES];
21. volatile bool setBufAReady = true;
22. volatile bool setBufBReady = false;
23. //频率相关数据
24. uint32\_t SAMP\_FREQ = 200\*1024;
25. uint32\_t systemClock = 120000000;
26. //无意义值
27. float f32\_nil=0;
28. int i32\_nil=0;
29. uint32\_t u32\_nil=0;
30. uint8\_t u8\_nil=0;
31. //调试信号
32. uint8\_t Debug\_Signal[10]={0};
33. //数据重装函数内置变量
34. uint32\_t Reload\_Index=0;
35. //步骤间传递变量
36. float Step3toStep4\_Basefreq=0;
37. //第二步
38. uint8\_t Step2\_Key\_Value=1;
39. //第x步
40. uint32\_t Stepx\_i1=0;
41. float Stepx\_Vout\_Buf[NUM\_SAMPLES]={0};
42. float Stepx\_Vmax=0,Stepx\_Vmin=0,Stepx\_Vpp=0;
43. //第三步
44. float Step3\_Feval=0;
45. uint32\_t Step3\_i1=0,Step3\_i2=0,Step3\_i3=0;
46. float Step3\_Vin[NUM\_SAMPLES]={0};
47. float Step3\_V\_Cplx[NUM\_SAMPLES\*2]={0};
48. float Step3\_Vout\_Mag[NUM\_SAMPLES\*2]={0};
49. float Step3\_Vout[NUM\_SAMPLES]={0};
50. float Step3\_Vmax=0;
51. uint32\_t Step3\_Vmax\_Index=0;
52. float Step3\_Basefreq=0;
53. float Step3\_Freqs[10]={0};
54. float Step3\_Fdif[10]={0};
55. uint32\_t Step3\_Fdif\_Max\_Index=0;
56. uint8\_t Step3\_Cir\_Times=0;
57. //第四步
58. uint32\_t Step4\_FS=200\*1024;
59. uint32\_t Step4\_i1=0,Step4\_i2=0,Step4\_i3=0;
60. float Step4\_Vin[NUM\_SAMPLES]={0};
61. float Step4\_V\_Cplx[NUM\_SAMPLES\*2]={0};
62. float Step4\_Vout\_Mag[NUM\_SAMPLES\*2]={0};
63. float Step4\_Vout[NUM\_SAMPLES]={0};
64. float Step4\_Vmax=0;
65. uint32\_t Step4\_Vmax\_Index=0;
66. float Step4\_Hal\_Wave[5]={0};
67. float Step4\_THD=0;

（2）函数声明

1. //自定义函数声明
2. //Init
3. void My\_Clock\_Init(void);
4. void My\_UART\_Init(void);
5. void My\_Timer\_Init(void);
6. void My\_ADC\_Init(void);
7. void My\_DMA\_Init(void);
8. void My\_Key\_Init(void);
9. void My\_Other\_Init(void);
10. void My\_Base\_Init(void);
11. void My\_Var\_Reload(void);
12. //Algorithm
13. float My\_Max(float a[],int b,int c);
14. float My\_Abs(float a);
15. void My\_FFT(float in\_buf[],float out\_buf[]);
16. void My\_FS\_Set(uint32\_t fre);
17. void My\_ADC\_Convert(void);
18. void My\_Delay\_ms(uint32\_t ms);
19. uint32\_t My\_Calculate\_FS(float freq);
20. //Interrupt
21. void ADC0SS2\_IRQHandler(void);

（3）算法函数

1. //算法函数
2. //Algorithm
3. float My\_Max(float a[],int b,int c)
4. {
5. float maxnum=0;
6. int i;
7. for(i=-(c-1)/2;i<(c+1)/2;i++)
8. {
9. if (a[b+i]>maxnum)
10. maxnum=a[b+i];
11. }
12. return maxnum;
13. }
14. float My\_Abs(float a)
15. {
16. if(a>=0) return a;
17. else return -a;
18. }
19. void My\_FFT(float in\_buf[],float out\_buf[])
20. {
21. float fft\_v\_buf[NUM\_SAMPLES\*2]={0.0f};
22. float fft\_mag\_buf[NUM\_SAMPLES\*2]={0.0f};
23. uint32\_t fft\_i=0;
24. for(fft\_i=0;fft\_i<NUM\_SAMPLES;fft\_i++)
25. {
26. fft\_v\_buf[fft\_i\*2]=in\_buf[fft\_i];
27. fft\_v\_buf[fft\_i\*2+1]=0.0f;
28. }
29. arm\_cfft\_f32(&arm\_cfft\_sR\_f32\_len1024,fft\_v\_buf,0,1);
30. arm\_cmplx\_mag\_f32(fft\_v\_buf,fft\_mag\_buf,NUM\_SAMPLES);
31. out\_buf[0]=fft\_mag\_buf[0]/(float)NUM\_SAMPLES;
32. for(fft\_i=1;fft\_i<NUM\_SAMPLES;fft\_i++)
33. {
34. out\_buf[fft\_i]=fft\_mag\_buf[fft\_i]/((float)NUM\_SAMPLES/2.0f);
35. }
36. }
37. void My\_FS\_Set(uint32\_t fre)
38. {
39. SAMP\_FREQ=fre;
40. MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, (systemClock/SAMP\_FREQ));
41. MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A);
42. }
43. void My\_ADC\_Convert(void)
44. {
46. while(!setBufAReady);//等待上次ADC转换完成
47. setBufAReady=false;
48. MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A);
49. while(!setBufAReady);//等待本次ADC转换完成
50. }
51. void My\_Delay\_ms(uint32\_t ms)
52. {
53. uint32\_t temp1=0,temp2=0;
55. for(temp2=0;temp2<ms;temp2++)
56. {
57. temp1=systemClock/12000;
58. while(--temp1);
59. }
60. }
61. uint32\_t My\_Calculate\_FS(float freq)
62. {
63. return (uint32\_t)freq\*100.0f/101.0f;
64. }

（4）主函数关键部分

1. //判断输入电压峰峰值并调整程控放大器
2. My\_FS\_Set(200\*1024);
3. {GPIOK->DATA&=~BIT3;GPIOK->DATA&=~BIT2;}
4. My\_Delay\_ms(5);
5. My\_ADC\_Convert();
6. for(Stepx\_i1=0;Stepx\_i1<NUM\_SAMPLES;Stepx\_i1++)
7. {
8. Stepx\_Vout\_Buf[Stepx\_i1]=(float)(dstBufferA[Stepx\_i1])\*3.3f/4096.0f;
9. }
10. arm\_max\_f32(Stepx\_Vout\_Buf,NUM\_SAMPLES,&Stepx\_Vmax,&u32\_nil);
11. arm\_min\_f32(Stepx\_Vout\_Buf,NUM\_SAMPLES,&Stepx\_Vmin,&u32\_nil);
12. Stepx\_Vpp=Stepx\_Vmax-Stepx\_Vmin;
13. if(Stepx\_Vpp<0.06f)
14. {GPIOK->DATA|=BIT2;GPIOK->DATA&=~BIT3;}
15. else if(Stepx\_Vpp<0.1f)
16. {GPIOK->DATA|=BIT3;GPIOK->DATA&=~BIT2;}
17. else if(Stepx\_Vpp<0.6f)
18. {GPIOK->DATA&=~BIT3;GPIOK->DATA&=~BIT2;}
19. //800kHz采样并做FFT变换，计算基波频率
20. My\_FS\_Set(800\*1024);
21. My\_ADC\_Convert();
22. for(Step3\_i1=0;Step3\_i1<NUM\_SAMPLES;Step3\_i1++)
23. {
24. Step3\_Vin[Step3\_i1]=(float)dstBufferA[Step3\_i1];
25. }
27. for(Step3\_i1=0;Step3\_i1<NUM\_SAMPLES;Step3\_i1++)
28. {
29. Step3\_V\_Cplx[Step3\_i1\*2]=Step3\_Vin[Step3\_i1];
30. Step3\_V\_Cplx[Step3\_i1\*2+1]=0.0f;
31. }
33. arm\_cfft\_f32(&arm\_cfft\_sR\_f32\_len1024,Step3\_V\_Cplx,0,1);
34. arm\_cmplx\_mag\_f32(Step3\_V\_Cplx,Step3\_Vout\_Mag,NUM\_SAMPLES);
35. Step3\_Vout[0]=Step3\_Vout\_Mag[0]/(float)NUM\_SAMPLES;
36. for(Step3\_i1=1;Step3\_i1<NUM\_SAMPLES;Step3\_i1++)
37. {
38. Step3\_Vout[Step3\_i1]=Step3\_Vout\_Mag[Step3\_i1]/((float)NUM\_SAMPLES/2.0f);
39. }
40. for(Step3\_i3=0;Step3\_i3<3;Step3\_i3++)
41. {
42. Step3\_Vout[Step3\_i3]=0;
43. }
44. arm\_max\_f32(Step3\_Vout,NUM\_SAMPLES/2,&Step3\_Vmax,&Step3\_Vmax\_Index);
46. Step3\_Basefreq=(float)Step3\_Vmax\_Index\*(float)SAMP\_FREQ/(float)NUM\_SAMPLES;
47. //根据基波频率调整等效采样频率，20kHz以下直接设置为200kHz采样率
48. Step4\_FS=My\_Calculate\_FS(Step3toStep4\_Basefreq);
49. My\_FS\_Set(Step4\_FS);
50. My\_ADC\_Convert();
51. //计算各谐波的归一化幅值并计算THD
52. Step4\_Hal\_Wave[0]=Step4\_Vout[Step4\_Vmax\_Index];
53. for(Step4\_i1=1;Step4\_i1<5;Step4\_i1++)
54. {
55. Step4\_Hal\_Wave[Step4\_i1]=My\_Max(Step4\_Vout,(int)Step4\_Vmax\_Index\*(Step4\_i1+1),5);
56. Step4\_Hal\_Wave[Step4\_i1]/=Step4\_Hal\_Wave[0];
57. }
58. Step4\_Hal\_Wave[0]=1;
59. Step4\_THD=100.0f\*sqrt(Step4\_Hal\_Wave[1]\*Step4\_Hal\_Wave[1]+Step4\_Hal\_Wave[2]\*Step4\_Hal\_Wave[2]+Step4\_Hal\_Wave[3]\*Step4\_Hal\_Wave[3]+Step4\_Hal\_Wave[4]\*Step4\_Hal\_Wave[4])/(Step4\_Hal\_Wave[0]);
60. //向屏幕以及ESP32发送信号分析结果
61. UARTStdioConfig(0, 115200, systemClock);
62. /\* 发送THD \*/
63. UARTprintf("x0.val=%d", (int)(thd\*100000.0f));
64. UARTprintf("%s", screen\_end);
65. UARTprintf("x0.vvs1=5");
66. UARTprintf("%s", screen\_end);
67. /\* 发送归一化幅值\*/
68. UARTprintf("x1.val=%d", (int)(up\_max[0]\*100000));
69. UARTprintf("%s", screen\_end);
70. UARTprintf("x1.vvs1=5");
71. UARTprintf("%s", screen\_end);
72. UARTprintf("x2.val=%d", (int)(up\_max[1]\*100000));
73. UARTprintf("%s", screen\_end);
74. UARTprintf("x2.vvs1=5");
75. UARTprintf("%s", screen\_end);
76. UARTprintf("x3.val=%d", (int)(up\_max[2]\*100000));
77. UARTprintf("%s", screen\_end);
78. UARTprintf("x3.vvs1=5");
79. UARTprintf("%s", screen\_end);
80. UARTprintf("x4.val=%d", (int)(up\_max[3]\*100000));
81. UARTprintf("%s", screen\_end);
82. UARTprintf("x4.vvs1=5");
83. UARTprintf("%s", screen\_end);
84. UARTprintf("x5.val=%d", (int)(up\_max[4]\*100000));
85. UARTprintf("%s", screen\_end);
86. UARTprintf("x5.vvs1=5");
87. UARTprintf("%s", screen\_end);
88. /\* 绘制波形曲线 \*/
89. for(wave\_count = 0; wave\_count < 250; wave\_count++){
90. wave[wave\_count] = up\_max[0]\*sin(wave\_count\*my\_PI/125) +
91. up\_max[1]\*sin(2\*wave\_count\*my\_PI/125) +
92. up\_max[2]\*sin(3\*wave\_count\*my\_PI/125) +
93. up\_max[3]\*sin(4\*wave\_count\*my\_PI/125) +
94. up\_max[4]\*sin(5\*wave\_count\*my\_PI/125);
95. }
96. for(wave\_count = 0; wave\_count < 250; wave\_count++){
97. UARTprintf("add s0.id,0,%d\xff\xff\xff",(int)(wave[wave\_count]\*50)+100);
98. }
99. UARTStdioConfig(2, 115200, systemClock);
100. UARTprintf("PKT %d %d %d %d %d %d %d 200000",(int)(thd\*100000.0f), (int)(basefreq), (int)(up\_max[0]\*100000),(int)(up\_max[1]\*100000),(int)(up\_max[2]\*100000),(int)(up\_max[3]\*100000),(int)(up\_max[4]\*100000));

## 附件7 ESP32部分代码

1. **void** uart2\_rx\_task(){
2. **while**(1){
3. memset(buff, 0, **sizeof**(buff));
4. **const** **int** len = uart\_read\_bytes(UART\_NUM\_2, buff, BUFF\_SIZE, 300/portTICK\_RATE\_MS);
6. **if**(len > 0){
7. send(client\_socket, buff, len, 0);
8. }
10. vTaskDelay(100/portTICK\_RATE\_MS);
11. }
12. }

15. **void** app\_main(**void**){
17. /\* 初始化NVS \*/
18. ESP\_ERROR\_CHECK(nvs\_flash\_init());
20. /\* 初始化WIFI和UART \*/
21. wifi\_init();
22. uart\_init();
24. /\* 初始化TCP \*/
25. tcp\_server\_init(&client\_socket);
27. /\* 创建uart2串口任务 \*/
28. xTaskCreate(uart2\_rx\_task, "uart2\_rx\_task", BUFF\_SIZE\*2, NULL, configMAX\_PRIORITIES, NULL);
29. ESP\_LOGI(TAG, "Create task done!\n");
30. }

## 附件8 APP 部分代码

1. **public** **class** MainActivity extends AppCompatActivity {
3. **public** **static** **float** PI = (**float**) 3.1415926;
5. /\*\*默认IP地址和端口号\*/
6. **public** **static** String IP\_address = null;
7. **public** **static** **int** St\_address = 0;
9. /\*\*Handler状态\*/
10. **public** **static** final **int** REFRESH = 1;
11. **public** **static** final **int** CONNECTED = 0;
13. /\*\*要绘制的数据\*/
14. **public** **static** **float**[] floats = **new** **float**[200];
16. /\*\*频率、有效值、峰峰值\*/
17. **public** **static** String thd, frq, u1, u2, u3, u4, u5, pp;
19. /\*\*缓冲区大小，单位: 字节\*/
20. **public** **static** final **int** BUFF\_SIZE = 2500;
22. /\*\*日志筛选\*/
23. **private** **static** final String TAG = "MainActivity";
25. @Override
26. **protected** **void** onCreate(Bundle savedInstanceState) {
27. super.onCreate(savedInstanceState);
28. setContentView(R.layout.activity\_main);
30. /\*\*初始化图像\*/
31. DrawLineChart chart = findViewById(R.id.chart);
32. chart.setMaxValue(10f);
33. chart.setMinValue(-10f);
34. chart.setNumberLine(5);
36. /\*\*初始化数据\*/
37. **for** (**int** i = 0; i < floats.length; i++) {
38. floats[i] = 0;
39. }
40. chart.setValue(floats);
41. getSupportActionBar().hide();
43. /\*\*跳转到设置界面\*/
44. Button set = (Button) findViewById(R.id.set);
45. set.setOnClickListener(**new** View.OnClickListener() {
46. @Override
47. **public** **void** onClick(View v) {
48. Intent intent  = **new** Intent(MainActivity.**this**, Connect.**class**);
49. startActivityForResult(intent, 1);
50. }
51. });
53. /\*\*点击进行连接和数据传输\*/
54. Button connect = (Button) findViewById(R.id.connect);
55. connect.setOnClickListener(**new** View.OnClickListener() {
56. @Override
57. **public** **void** onClick(View v) {
59. /\*\*若未填写信息则弹出警告\*/
60. **if**(IP\_address == null || St\_address == 0){
61. Toast.makeText(MainActivity.**this**, "请先点击设置", Toast.LENGTH\_SHORT).show();
62. }
64. **else** {
65. **new** ConnectThread().start();
66. }
67. }
68. });
69. }

72. /\*\*创建Handler用于更新曲线\*/
73. **private** Handler handler = **new** Handler(){
74. @Override
75. **public** **void** handleMessage(Message msg) {
76. **if**(msg.what == REFRESH){
77. /\*\*更新曲线\*/
78. DrawLineChart chart = findViewById(R.id.chart);
80. chart.setMinValue((-1) \* getNum(pp));
81. chart.setMaxValue(getNum(pp));
82. chart.setNumberLine(5);
83. chart.setValue(floats);
84. chart.requestLayout();
86. /\*\*更新THD\*/
87. TextView THD = (TextView)findViewById(R.id.THD);
88. THD.setText("THD: " + thd + "%");
90. /\*\*更新频率\*/
91. TextView FRQ = (TextView)findViewById(R.id.FRQ);
92. FRQ.setText("Frq: " + frq + "Hz");
94. /\*\*更新归一化幅值\*/
95. TextView U1 = (TextView)findViewById(R.id.U1);
96. U1.setText("U1: " + u1);
98. TextView U2 = (TextView)findViewById(R.id.U2);
99. U2.setText("U2: " + u2);
101. TextView U3 = (TextView)findViewById(R.id.U3);
102. U3.setText("U3: " + u3);
104. TextView U4 = (TextView)findViewById(R.id.U4);
105. U4.setText("U4: " + u4);
107. TextView U5 = (TextView)findViewById(R.id.U5);
108. U5.setText("U5: " + u5);
109. }
111. **else** **if**(msg.what == CONNECTED){
112. Toast.makeText(MainActivity.**this**, "已连接", Toast.LENGTH\_SHORT).show();
113. }
114. }
115. };

118. /\*\*连接线程\*/
119. **private** **class** ConnectThread extends Thread{
120. @Override
121. **public** **void** run() {
122. Message msg = **new** Message();
124. **try** {
125. Socket s = **new** Socket(IP\_address, St\_address);
126. InputStream is = s.getInputStream();
127. OutputStream os = s.getOutputStream();
128. byte[] b = **new** byte[BUFF\_SIZE];
130. /\*\*成功连接\*/
131. msg.what = CONNECTED;
132. handler.sendMessage(msg);
134. /\*\*发送ID码验证身份\*/
135. //os.write("Phone".getBytes());
137. /\*\*线程阻塞\*/
138. **while** (**true**){
139. // 读取信息
140. **int** len = is.read(b);
141. String message = **new** String(b, 0, len);
142. Log.d(TAG, "Received Data:" + message);


146. // 分割数据流
147. String[] data = message.split(" ");
148. Log.d(TAG, "DataLength:" + data.length);
150. **if**(data[0].equals("PKT") && data.length == 9){
151. // 提取THD、归一化幅值
152. thd = String.valueOf(Float.parseFloat(data[1])/1000);
153. frq = data[2]+"00";
154. Log.i(TAG, "FRQ:" + frq);
155. u1 = String.valueOf(Float.parseFloat(data[3]));
156. u2 = String.valueOf(Float.parseFloat(data[4])/100000);
157. u3 = String.valueOf(Float.parseFloat(data[5])/100000);
158. u4 = String.valueOf(Float.parseFloat(data[6])/100000);
159. u5 = String.valueOf(Float.parseFloat(data[7])/100000);
160. pp = String.valueOf(Float.parseFloat(data[8]));
162. // 提取波形信息
163. **for**(**int** i = 0; i < 200; i++){
164. floats[i] = (**float**) (Float.parseFloat(u1)\*(Math.sin(i\*PI/100)))    +
165. (**float**) (Float.parseFloat(u2)\*(Math.sin(2\*i\*PI/100)))  +
166. (**float**) (Float.parseFloat(u3)\*(Math.sin(3\*i\*PI/100)))  +
167. (**float**) (Float.parseFloat(u4)\*(Math.sin(4\*i\*PI/100)))  +
168. (**float**) (Float.parseFloat(u5)\*(Math.sin(5\*i\*PI/100)));
169. floats[i] = floats[i];
170. }
172. /\*\*更新UI\*/
173. Message msg0 = **new** Message();
174. msg0.what = REFRESH;
175. handler.sendMessage(msg0);
176. }
177. }
178. } **catch** (IOException e) {
179. e.printStackTrace();
180. }
181. }
182. }
184. /\*\*接受设置界面传递的参数\*/
185. @Override
186. **protected** **void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {
188. **if**(data != null){
189. super.onActivityResult(requestCode, resultCode, data);
191. /\*\*获取IP地址和端口号\*/
192. **if** (requestCode == 1) {
193. IP\_address = data.getStringExtra("IP\_address");
194. St\_address = data.getIntExtra("St\_address", 0);
195. }
196. }
198. }
200. /\*\*将字符串中的数字提取出来\*/
201. **public** **float** getNum(String str){
202. **int** index = str.length();
204. **for**(**int** i = 0; i < str.length(); i++){
205. **if**(str.charAt(i) > '9'){
206. index = i;
207. **break**;
208. }
209. }
211. **return** Float.parseFloat(str.substring(0, index));
212. }
213. }