**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、信号波形和信号归一化幅值。经测试，本设计功能齐全，测量、显示、失真度计算等均达到和超过指标要求。

【关键词】MSP432E401Y；等效采样；FFT变换；频谱搬移；手机显示

**目 录**

[信号失真度测量装置 1](#_Toc87893770)

[1 设计任务与要求 1](#_Toc87893771)

[1.1设计任务 1](#_Toc87893772)

[1.2设计要求 1](#_Toc87893773)

[2 方案设计与论证 2](#_Toc87893774)

[2.1信号采样 2](#_Toc87893775)

[2.2 FFT数据类型对比 3](#_Toc87893776)

[3 理论分析与计算 3](#_Toc87893777)

[3.1 信号处理 3](#_Toc87893778)

[3.1.1 信号放大电路 3](#_Toc87893779)

[3.1.2 加法器电路 4](#_Toc87893780)

[3.2输入信号失真度分析与计算 4](#_Toc87893781)

[3.2.1 信号的非线性失真程度 4](#_Toc87893782)

[3.2.2 归一化幅值 4](#_Toc87893783)

[3.3 FFT变换 4](#_Toc87893784)

[3.3.1 窗函数 4](#_Toc87893785)

[3.3.2 频谱分辨率 5](#_Toc87893786)

[3.4 等效采样 5](#_Toc87893787)

[4 系统软、硬件设计 6](#_Toc87893788)

[4.1 系统硬件设计 6](#_Toc87893789)

[4.1.1 电源 6](#_Toc87893790)

[4.1.2 信号处理电路 7](#_Toc87893791)

[4.1.3 主控MCU 7](#_Toc87893792)

[4.1.4 无线传输模块 7](#_Toc87893793)

[4.2 系统软件设计 7](#_Toc87893794)

[4.2.1 MSP432主程序 7](#_Toc87893795)

[5 系统测试及结果 9](#_Toc87893796)

[5.1 测试仪器 9](#_Toc87893797)

[5.2 测试方法 9](#_Toc87893798)

[5.3 性能指标 9](#_Toc87893799)

[6 总结与展望 9](#_Toc87893800)

[附 件 11](#_Toc87893801)

[附件1 电源电路原理图 11](#_Toc87893802)

[附件2 程控放大器电路原理图 11](#_Toc87893803)

[附件3 MSP432代码 12](#_Toc87893804)

[附件4 ESP32部分代码 35](#_Toc87893805)

[附件5 安卓上位机部分代码 41](#_Toc87893806)

## 1 设计任务与要求

### 1.1设计任务

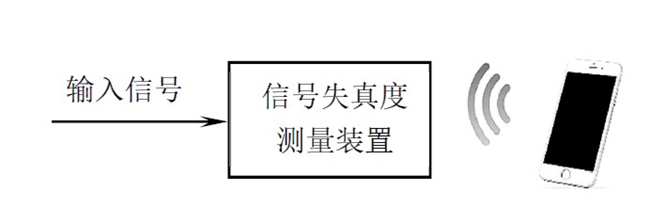
设计制作信号失真度测量装置，对来自函数/任意波形发生器的周期信号（以下简称为输入信号）进行采集分析，测得输入信号的总谐波失真*THD*（以下简称为失真度），并可在手机上显示测量信息。测量装置系统组成示意图如图1 所示。

图1 测量装置系统组成示意图

### 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

## 2 方案设计与论证

系统整体框架如图所示：

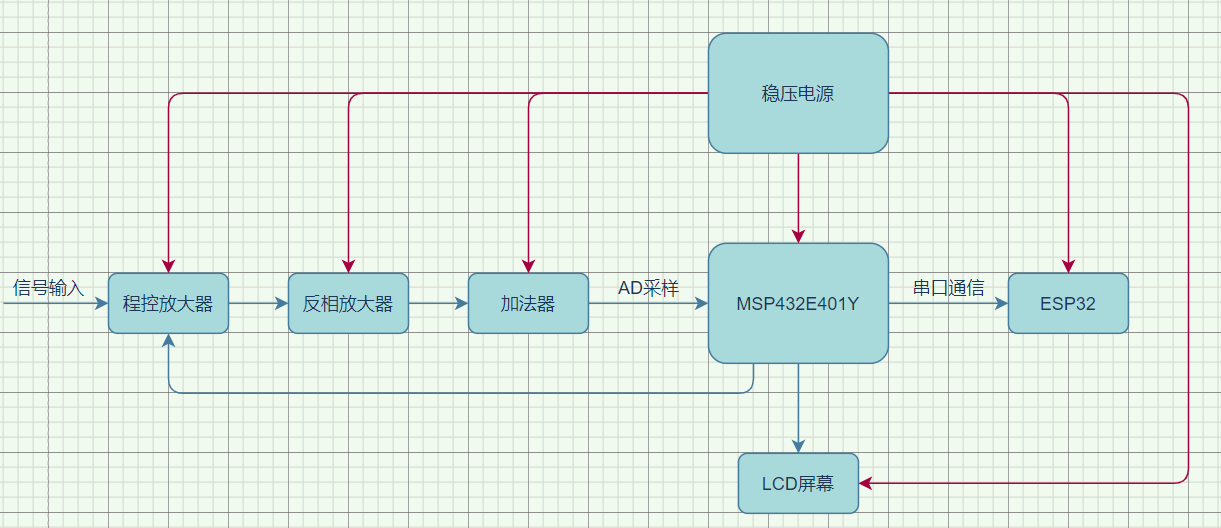


图1 信号失真度测量装置系统框图

### 2.1信号采样

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。

方案一：直接采样。

以1M SPS固定采样率进行采样，对采集到的数据直接进行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，能够直接对输入信号进行采样和计算；缺点：频谱分辨率较低，误差较大。

方案二：等效采样。

以200K SPS固定采样率进行采样，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，最后对恢复出的原始信号进行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，频谱分辨率有所提高；缺点：采样时间有所增加，测量时间变长，软件开发难度较大，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。

### 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。

## 3 理论分析与计算

### 3.1 信号处理

3.1.1 信号放大电路

由函数发生器输出的信号峰峰值范围为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地放大。本设计使用程控放大器和固定增益的反相放大器组成信号放大电路，程控放大器可选倍率x1，x2，x5，x10，其根据输入信号的幅值调整自身放大倍数，确保最终的的信号处于ADC输入范围内。

由于程控放大器使用仪表放大器INA821级联而成，其压摆率较低(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。该反向放大器使用OPA140搭建，增益为4，该增益下-3dB带宽约为2Mhz。由程控放大器和反相放大器组成的放大电路最终方案如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
| **输入信号** | **程控放大倍数** | **总放大倍数** | **输出信号** |
| 30mVpp – 60mVpp | 10 | 40 | 1.2Vpp – 2.4Vpp |
| 60mVpp – 120mVpp | 5 | 20 | 1.2Vpp – 2.4Vpp |
| 120mVpp – 300mVpp | 2 | 8 | 0.96Vpp – 2.4Vpp |
| 300mVpp – 600mVpp | 1 | 4 | 1.2Vpp – 2.4Vpp |

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。

### 3.2输入信号失真度分析与计算

3.2.1 信号的非线性失真程度

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*（total harmonic distortion）”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：

本题信号失真度测量采用近似方式，测量和分析输入信号谐波成分时，限定

只处理到5次谐波。定义

为本题失真度的标称值。

若失真度测量值为THDx ，则失真度测量误差的绝对值为

3.2.2 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、、、……。

### 3.3 FFT变换

3.3.1 窗函数

假设采样频率为Fs，采样点数为N，那么FFT的结果就是一个长度为N的复数数组。 数组的每一个点就对应着一个频率点，这个点的模值，就是该频率值对应的的幅度特性。假设原始信号某一频率分量的峰值为A，那么FFT结果的每个点（除了第一个点直流分量之外）的模值就是A的N/2倍。第一个点就是直流分量，它的模值是直流分量的N倍。这样，我们就能通过FFT的结果计算得到原始信号各个频率分量的幅值。

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。

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。

### 3.4 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，在电路上只对取样前的电路具有高频的要求，大大降低采样电路和采样变换后的信号处理对速度的要求，简化了整个系统的设计难度。若采用实时采样技术，对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。

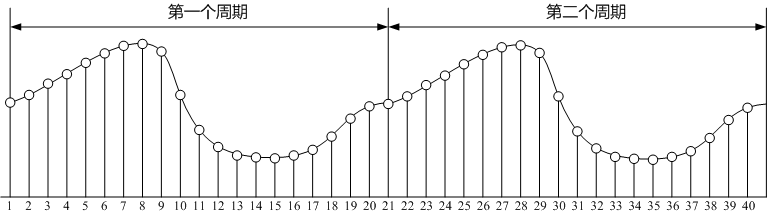


图3 两个信号周期连续实时采样获取的采样点

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奇数的10个采样点（1、3、5……19）和第二个周期中序号为偶数的10个采样点（22、24、26……40），再按照图5的顺序排列。

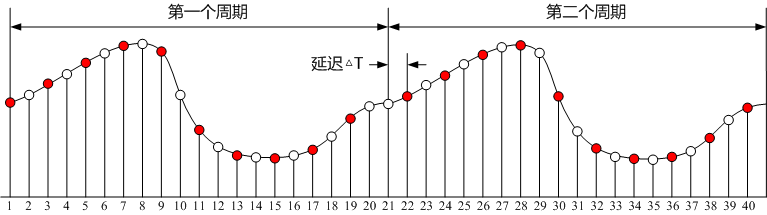


图4 周期信号等效采样原理

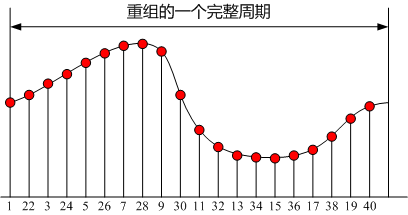


图5 采用等效采样技术重组的一个完整信号周期

这种采样方式，我们称为等效采样方式。等效采样降低了实际采样速率，减小了单位时间内的采样点数，降低了对数据存储速度和容量的要求。

## 4 系统软、硬件设计

### 4.1 系统硬件设计

4.1.1 电源

本设计采用LM317和LM337设计可调稳压电源，同时对输出电压进行滤波操作。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。固定增益的反相放大器由 OPA140搭建。相关原理图见附件2.

加法器电路使用电容和电阻网络构成，原理如图6所示。由于其输入端等效为一个高通滤波器，而输入信号范围高于1kHz，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。

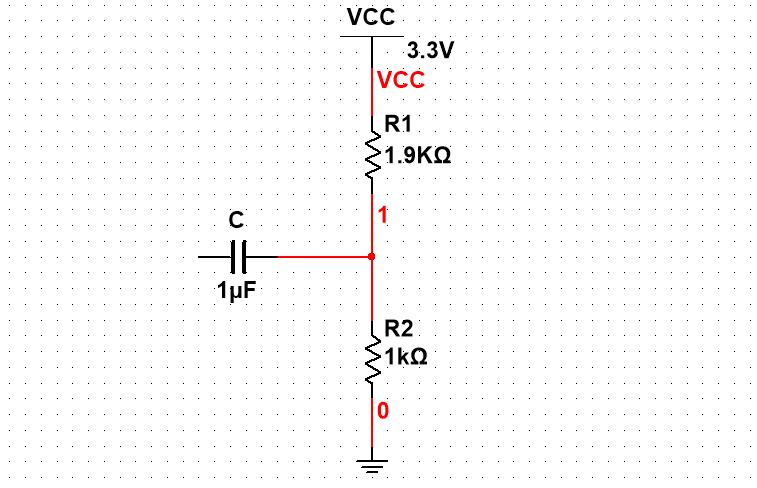


图6 加法器电路

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。经调试，模块的采样率可以达到1.2M SPS，能够精确地测量1kHz - 100kHz的信号。

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，速度高达150 Mbps。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。

### 4.2 系统软件设计

4.2.1 MSP432主程序

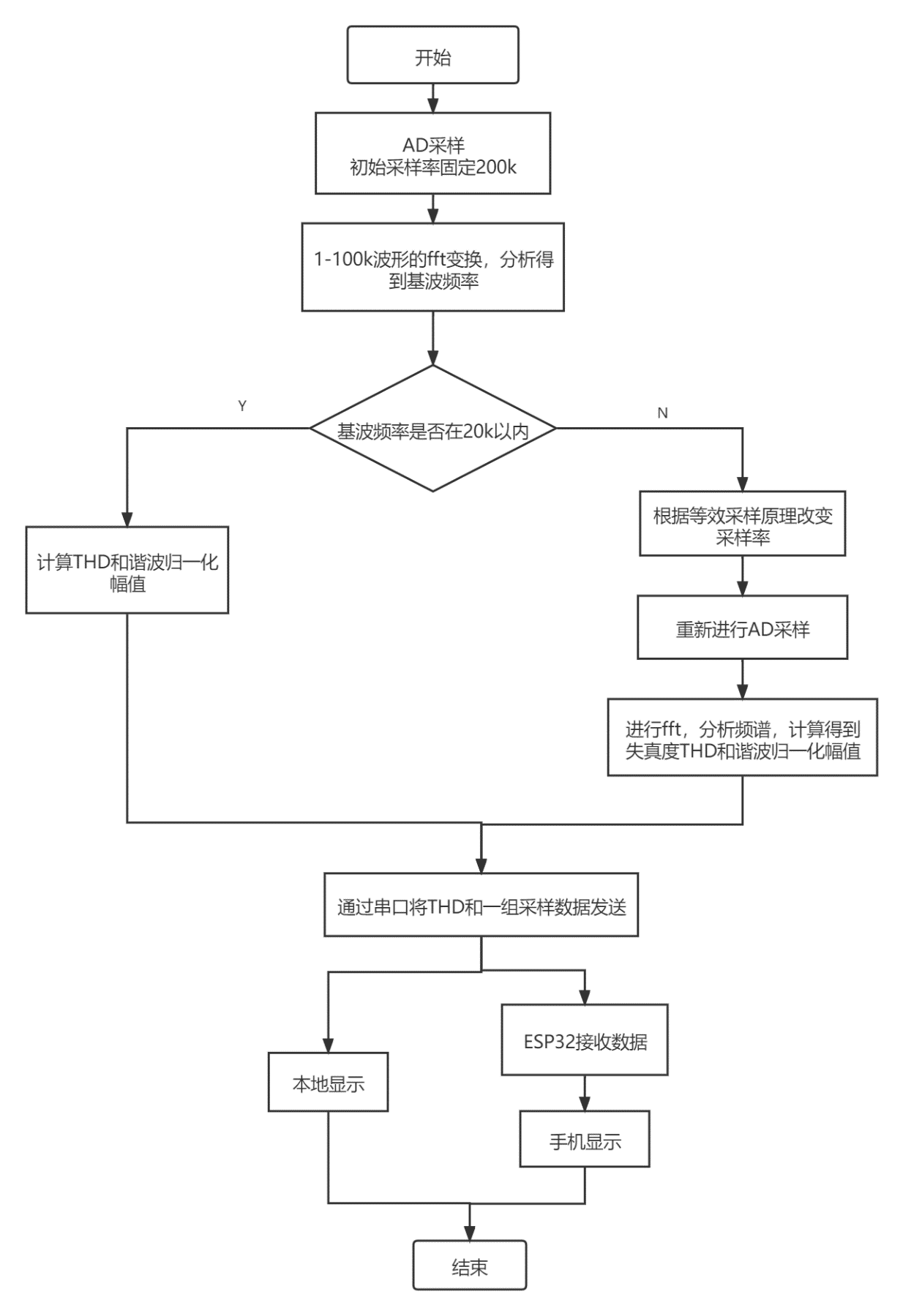


图6 主程序流程图

由于基础设定ADC的采样率为200kHz，故当输入信号的基波频率小于20kHz时不需要进行重新采样，可以直接分析FFT变换后的信号频谱，得到基波频率和失真度THD。若输入信号的基波频率高于20kHz时，则需要根据FFT变换得到的基波频率，改变MSP432E401Y单片机片内ADC的采样率，新的采样率为输入信号基波频率的10倍。最后将计算得到的THD，幅值归一化和一个周期内的波形等数据通过串口发送给本地显示的串口屏和ESP32，ESP32将数据通过WIFI发送到手机，实现无线传输和显示测量结果。

## 5 系统测试及结果

### 5.1 测试仪器

表1：测试仪器

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

### 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读出测量出的非线性失真程度，记录数据。

### 5.3 性能指标

1. 非线性失真度测量值的计算：；
2. 谐波归一化幅值为1、***(Um2/Um1)、(Um3/Um2)***…

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| THD标称值 |  |  |  |  |
| THD测量值 |  |  |  |  |
| Δ |  |  |  |  |

表2 THD测量精度

## 6 总结与展望

本系统以MSP432E401Y单片机为核心部分，根据单片机片内ADC采集到的数据，经计算得出输入信号的非线性失真程度。在系统设计中，力求硬件线路简单，充分发挥软件编程灵活的特点，来满足系统设计的要求。

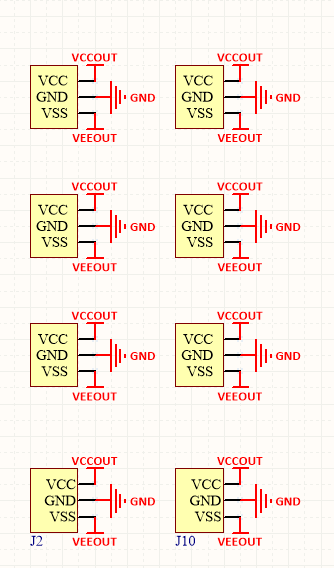
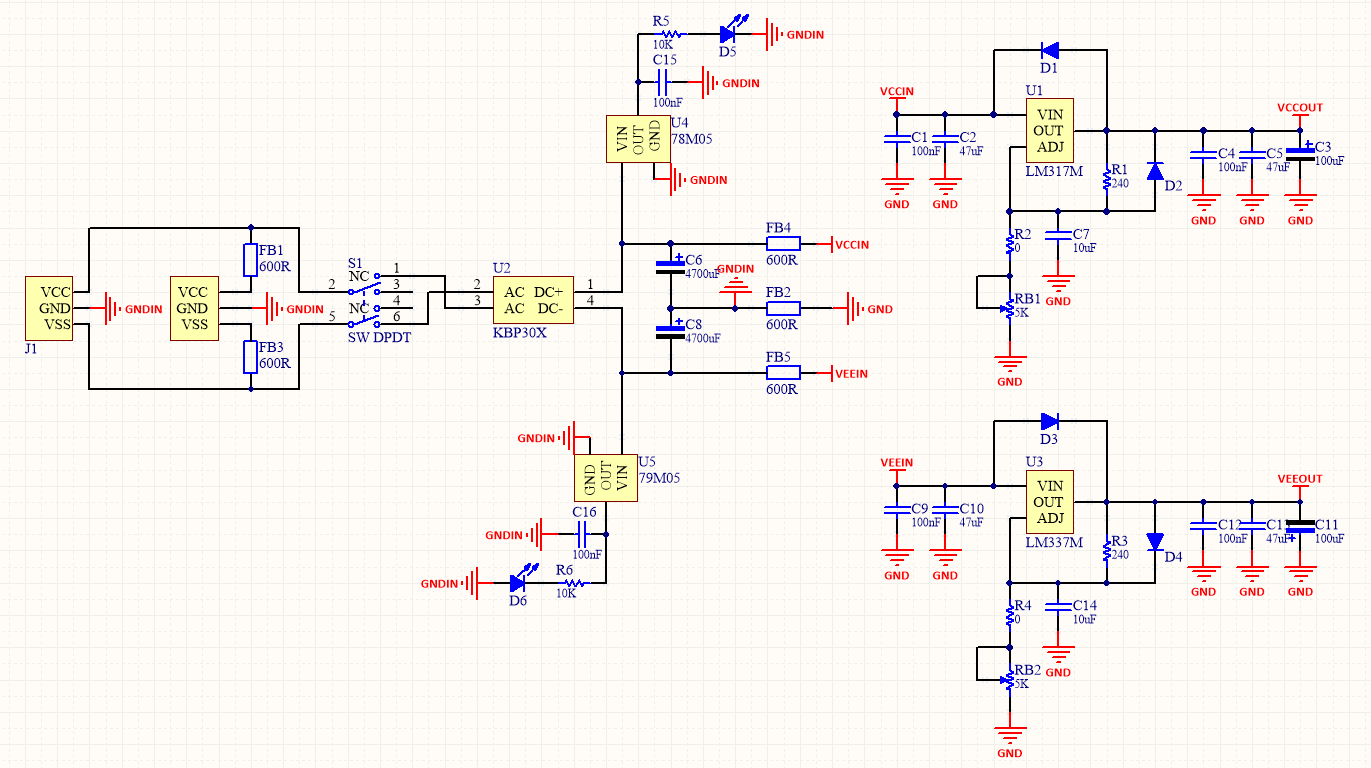
从测试结果看，本设计完全完成了题目的要求，信号失真度测量精度高，误差小，可以通过较低采样率的片内ADC对高频信号进行采样。系统要求测量失真度能够精确达到指标要求，测量数据的分辨率都到达要求；另外增加屏幕显示使得数据清晰明了。

在本次设计中我们创新性地设计了由低采样率地ADC采集高频信号的算法，若能够将这种算法进一步优化和推广，在未来的工程实践和工业产品设计中都将会有很广泛的实际应用。

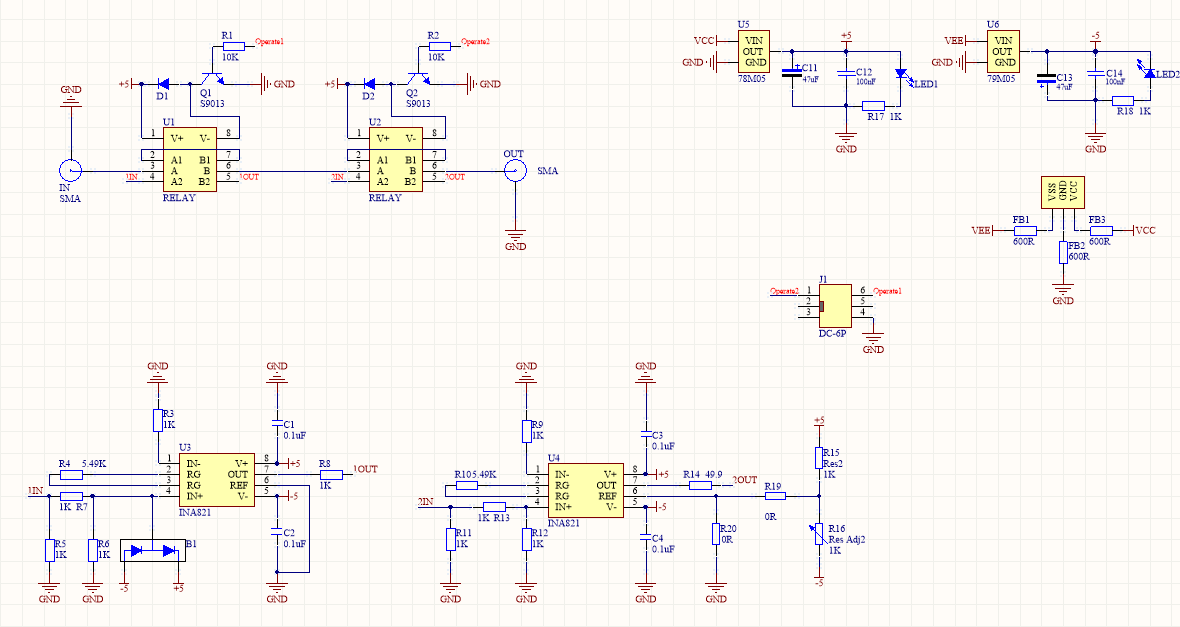
## 

## 附 件

### 附件1 电源电路原理图



### 附件2 程控放大器电路原理图



### 附件3 MSP432代码

1. #include <ti/devices/msp432e4/driverlib/driverlib.h>
2. #include <stdio.h>
3. #include <stdint.h>
4. #include <stdlib.h>
5. #include <stdbool.h>
6. #include "oled.h"
7. #include <math.h>
8. #include "arm\_math.h"
9. #include "uartstdio.h"
10. #include "arm\_const\_structs.h"

13. #define NUM\_SAMPLES 1024
14. #define PI 3.1415926
15. #define IFFTFLAG   0
16. #define BITREVERSE 1
18. **static** uint32\_t dstBufferA[NUM\_SAMPLES];
19. **static** uint32\_t dstBufferB[NUM\_SAMPLES];
20. **volatile** **bool** setBufAReady = **false**;
21. **volatile** **bool** setBufBReady = **false**;
22. uint32\_t SAMP\_FREQ = 32000;

25. **char** screen\_end[3] = {0xFF, 0xFF, 0xFF};
26. **char** esp32\_send[64];
27. **float** fre[1024]={0};
28. **float** temp\_maxv;
29. **float** old\_temp\_maxv;
30. uint32\_t temp\_maxv\_index;
31. **int** fft1\_i1,fft1\_i2;
32. **float** temp\_c\_buf[NUM\_SAMPLES\*2]={0};
33. **float** temp\_in\_c\_buf[NUM\_SAMPLES\*2]={0};
34. **float** temp\_mag\_buf[NUM\_SAMPLES\*2]={0};
35. **float** true\_vout[NUM\_SAMPLES]={0};
36. **float** true\_fout[NUM\_SAMPLES]={0};
38. **float** best\_vout\_buf[NUM\_SAMPLES];
39. **float** best\_fout\_buf[NUM\_SAMPLES];
41. **float** wave[250];
42. uint32\_t wave\_count;
44. **float** voutmax;
45. **float** step2\_vinmax,step2\_vinmin,step2\_vpp;
46. uint32\_t step2\_vinmax\_index,step2\_vinmin\_index;
47. **float** fft\_f\_out\_finnal\_value[5];
48. **float** step1\_basefre;
49. uint8\_t step1\_circle\_i1;
50. uint8\_t step1\_circle\_i2;

53. **float** fft\_winout\_sampling[NUM\_SAMPLES];
54. uint32\_t voutmax\_index;
56. **int** button\_value,cir\_times=1;
58. **float** fftout\_max\_info[5]={0};
59. **volatile** int16\_t fftOutput[NUM\_SAMPLES\*2];
61. **volatile** float32\_t rmsBuff;
62. **volatile** float32\_t dcBuff;
63. float32\_t rmsCalculation;
65. uint32\_t low\_freq=0;
66. uint32\_t step2\_vol\_index;
67. **float** step2\_vol\_list[NUM\_SAMPLES];
68. uint8\_t step2\_amp\_index;
69. **float** up\_freq=0;
70. **float** up\_max[5]={0};
71. **float** low\_max[5]={0};
72. **int** max\_index[5]={0};
73. **float** thd;
74. **float** basefreq;
76. **char** screen\_buf[100];
77. uint32\_t trans\_upmax[5]={0};
78. uint32\_t trans\_thd;
79. uint32\_t trans\_wavebuf[1024];
81. #if defined(\_\_ICCARM\_\_)
82. #pragma data\_alignment=1024
83. uint8\_t pui8ControlTable[1024];
84. #elif defined(\_\_TI\_ARM\_\_)
85. #pragma DATA\_ALIGN(pui8ControlTable, 1024)
86. uint8\_t pui8ControlTable[1024];
87. #else
88. uint8\_t pui8ControlTable[1024] \_\_attribute\_\_ ((aligned(1024)));
89. #endif

92. **void** timer\_fre\_set(uint32\_t fre,uint32\_t systemClock)
93. {
94. SAMP\_FREQ=fre;
95. MAP\_TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_A\_PERIODIC);
96. MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, (systemClock/SAMP\_FREQ));
97. MAP\_TimerControlTrigger(TIMER0\_BASE, TIMER\_A, **true**);
98. MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A);
99. }
101. **void** start\_adc\_convert(**void**)
102. {
103. **while**(!setBufAReady);
104. MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A);
105. setBufAReady = **false**;
106. }
108. uint32\_t calculate\_FS(**float** freq)
109. {
110. **return** (uint32\_t)(freq/1024-1) \* 1024;
111. }
113. **float** \*my\_fft\_f(**float** \*sampling\_buf,unsigned **int** cir\_times,unsigned **int** point\_num)
114. {
115. **int** fft1\_i1,fft1\_i2;
116. **float** temp\_c\_buf[**sizeof**(sampling\_buf)\*2]={0};
117. **float** temp\_in\_c\_buf[**sizeof**(sampling\_buf)\*2]={0};
118. **float** temp\_mag\_buf[**sizeof**(sampling\_buf)\*2]={0};
119. **float** \*big\_buf;
120. arm\_cfft\_instance\_f32 \* S;
121. **switch**(point\_num)
122. {
123. **case** 16:
124. \*S=arm\_cfft\_sR\_f32\_len16;
125. **break**;
126. **case** 32:
127. \*S=arm\_cfft\_sR\_f32\_len32;
128. **break**;
129. **case** 64:
130. \*S=arm\_cfft\_sR\_f32\_len64;
131. **break**;
132. **case** 128:
133. \*S=arm\_cfft\_sR\_f32\_len128;
134. **break**;
135. **case** 256:
136. \*S=arm\_cfft\_sR\_f32\_len256;
137. **break**;
138. **case** 512:
139. \*S=arm\_cfft\_sR\_f32\_len512;
140. **break**;
141. **case** 1024:
142. \*S=arm\_cfft\_sR\_f32\_len1024;
143. **break**;
144. **case** 2048:
145. \*S=arm\_cfft\_sR\_f32\_len2048;
146. **break**;
147. **case** 4096:
148. \*S=arm\_cfft\_sR\_f32\_len4096;
149. **break**;
150. **default**:
151. UARTprintf("FFT param error!\n");
152. **break;**
153. **}**
154. big\_buf=malloc(**sizeof**(sampling\_buf)\*cir\_times);
155. **for**(fft1\_i1=0;fft1\_i1<**sizeof**(sampling\_buf);fft1\_i1++)
156. {
157. temp\_c\_buf[fft1\_i1\*2]=sampling\_buf[fft1\_i1];
158. }
160. arm\_copy\_f32(temp\_c\_buf,temp\_in\_c\_buf,**sizeof**(sampling\_buf)\*2);
161. **for**(fft1\_i2=0;fft1\_i2<cir\_times;fft1\_i2++)
162. {
163. **for**(fft1\_i1=0;fft1\_i1<**sizeof**(sampling\_buf);fft1\_i1++)
164. {
165. temp\_c\_buf[fft1\_i1\*2]=temp\_in\_c\_buf[fft1\_i1\*2]\*arm\_cos\_f32(-fft1\_i1\*(PI/(**float**)(point\_num\*cir\_times)));
166. temp\_c\_buf[fft1\_i1\*2+1]=temp\_in\_c\_buf[fft1\_i1\*2]\*arm\_sin\_f32(-fft1\_i1\*(PI/(**float**)(point\_num\*cir\_times)));
167. }
169. arm\_cfft\_f32(S,temp\_c\_buf,0,1);
170. arm\_cmplx\_mag\_f32(temp\_c\_buf,temp\_mag\_buf,point\_num);
172. **for**(fft1\_i1=0;fft1\_i1<**sizeof**(sampling\_buf);fft1\_i1++)
173. {
174. big\_buf[fft1\_i2+fft1\_i1\*cir\_times]=temp\_c\_buf[fft1\_i1];
175. }
176. }
177. **return** big\_buf;
178. }
179. **void** ADC0SS2\_IRQHandler(**void**)
180. {
181. uint32\_t getIntStatus;
182. uint32\_t getDMAStatus;
184. /\* Get the interrupt status from the ADC \*/
185. getIntStatus = MAP\_ADCIntStatusEx(ADC0\_BASE, **true**);
187. /\* Clear the ADC interrupt flag. \*/
188. MAP\_ADCIntClearEx(ADC0\_BASE, getIntStatus);
190. /\* Read the primary and alternate control structures to find out which
191. \* of the structure has completed and generated the done interrupt. Then
192. \* re-initialize the appropriate structure \*/
193. getDMAStatus = MAP\_uDMAChannelModeGet(UDMA\_CH16\_ADC0\_2 |
194. UDMA\_PRI\_SELECT);
196. /\* Check if the primary or alternate channel has completed. On completion
197. \* re-initalize the channel control structure. If the Primary channel has
198. \* completed then set Buffer-A ready flag so that the main application
199. \* may perform the DSP computation. Similarly if the Alternate channel
200. \* has completed then set Buffer-B ready flag so that the main application
201. \* may perform the DSP computation. \*/
202. **if**(getDMAStatus == UDMA\_MODE\_STOP)
203. {
204. MAP\_uDMAChannelTransferSet(UDMA\_CH16\_ADC0\_2 | UDMA\_PRI\_SELECT,
205. UDMA\_MODE\_PINGPONG,
206. (**void** \*)&ADC0->SSFIFO2, (**void** \*)&dstBufferA,
207. **sizeof**(dstBufferA)/4);
208. setBufAReady = **true**;
209. MAP\_TimerDisable(TIMER0\_BASE, TIMER\_A);
210. }
212. getDMAStatus = MAP\_uDMAChannelModeGet(UDMA\_CH16\_ADC0\_2 |
213. UDMA\_ALT\_SELECT);
215. **if**(getDMAStatus == UDMA\_MODE\_STOP)
216. {
217. MAP\_uDMAChannelTransferSet(UDMA\_CH16\_ADC0\_2 | UDMA\_ALT\_SELECT,
218. UDMA\_MODE\_PINGPONG,
219. (**void** \*)&ADC0->SSFIFO2, (**void** \*)&dstBufferB,
220. **sizeof**(dstBufferB)/4);
221. setBufBReady = **true**;
222. }
223. }
225. **void** ConfigureUART(uint32\_t systemClock)
226. {
227. /\* Enable the clock to GPIO port A and UART 0 \*/
228. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);
229. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);
231. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART2);
232. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOD);
234. /\* Configure the GPIO Port A for UART 0 \*/
235. MAP\_GPIOPinConfigure(GPIO\_PA0\_U0RX);
236. MAP\_GPIOPinConfigure(GPIO\_PA1\_U0TX);
237. MAP\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);
239. MAP\_GPIOPinConfigure(GPIO\_PD4\_U2RX);
240. MAP\_GPIOPinConfigure(GPIO\_PD5\_U2TX);
241. MAP\_GPIOPinTypeUART(GPIO\_PORTD\_BASE, GPIO\_PIN\_4 | GPIO\_PIN\_5);
243. /\* Configure the UART for 115200 bps 8-N-1 format \*/
244. UARTStdioConfig(0, 115200, systemClock);
245. UARTStdioConfig(2, 115200, systemClock);
246. }
248. **int** main(**void**)
249. {
250. **char** buf[32];
251. uint32\_t systemClock;
252. uint32\_t t,i;
254. /\* Configure the system clock for 120 MHz \*/
255. systemClock = MAP\_SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ |
256. SYSCTL\_OSC\_MAIN |
257. SYSCTL\_USE\_PLL |
258. SYSCTL\_CFG\_VCO\_480), 120000000);
260. /\* Initialize serial console \*/
261. ConfigureUART(systemClock);
262. /\* Enable the clock to GPIO Port E and wait for it to be ready \*/
263. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);
264. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOK);
265. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOJ);
266. **while**(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOJ)))
267. {
268. }
270. /\* Configure PE3 as ADC input channel \*/
271. MAP\_GPIOPinTypeADC(GPIO\_PORTE\_BASE, GPIO\_PIN\_3);
272. GPIOJ->DIR&=~BIT0;
273. GPIOJ->PUR|=BIT0;
274. GPIOJ->DEN|=BIT0;
275. GPIOJ->DATA|=BIT0;
276. GPIOK->DIR|=BIT2+BIT3;
277. GPIOK->DEN|=BIT2+BIT3+BIT4+BIT5;
278. OLED\_Init();
279. OLED\_Clear();
280. /\* Enable the clock to ADC-0 and wait for it to be ready \*/
281. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);
282. **while**(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_ADC0)))
283. {
284. }
286. /\* Configure Sequencer 2 to sample the analog channel : AIN0. The end of
287. \* conversion and interrupt generation is set for AIN0 \*/
288. MAP\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_CH0 | ADC\_CTL\_IE |
289. ADC\_CTL\_END);
291. /\* Enable sample sequence 2 with a timer signal trigger.  Sequencer 2
292. \* will do a single sample when the timer generates a trigger on timeout\*/
293. MAP\_ADCSequenceConfigure(ADC0\_BASE, 2, ADC\_TRIGGER\_TIMER, 2);
295. /\* Clear the interrupt status flag before enabling. This is done to make
296. \* sure the interrupt flag is cleared before we sample. \*/
297. MAP\_ADCIntClearEx(ADC0\_BASE, ADC\_INT\_DMA\_SS2);
298. MAP\_ADCIntEnableEx(ADC0\_BASE, ADC\_INT\_DMA\_SS2);
300. /\* Enable the DMA request from ADC0 Sequencer 2 \*/
301. MAP\_ADCSequenceDMAEnable(ADC0\_BASE, 2);
303. /\* Since sample sequence 2 is now configured, it must be enabled. \*/
304. MAP\_ADCSequenceEnable(ADC0\_BASE, 2);
306. /\* Enable the Interrupt generation from the ADC-0 Sequencer \*/
307. MAP\_IntEnable(INT\_ADC0SS2);
309. /\* Enable the DMA and Configure Channel for TIMER0A for Ping Pong mode of
310. \* transfer \*/
311. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UDMA);
312. **while**(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_UDMA)))
313. {
314. }
316. MAP\_uDMAEnable();
318. /\* Point at the control table to use for channel control structures. \*/
319. MAP\_uDMAControlBaseSet(pui8ControlTable);
321. /\* Map the ADC0 Sequencer 2 DMA channel \*/
322. MAP\_uDMAChannelAssign(UDMA\_CH16\_ADC0\_2);
324. /\* Put the attributes in a known state for the uDMA ADC0 Sequencer 2
325. \* channel. These should already be disabled by default. \*/
326. MAP\_uDMAChannelAttributeDisable(UDMA\_CH16\_ADC0\_2,
327. UDMA\_ATTR\_ALTSELECT | UDMA\_ATTR\_USEBURST |
328. UDMA\_ATTR\_HIGH\_PRIORITY |
329. UDMA\_ATTR\_REQMASK);
331. /\* Configure the control parameters for the primary control structure for
332. \* the ADC0 Sequencer 2 channel. The primary control structure is used for
333. \* copying the data from ADC0 Sequencer 2 FIFO to dstBufferA. The transfer
334. \* data size is 32 bits and the source address is not incremented while
335. \* the destination address is incremented at 32-bit boundary.
336. \*/
337. MAP\_uDMAChannelControlSet(UDMA\_CH16\_ADC0\_2 | UDMA\_PRI\_SELECT,
338. UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE |
339. UDMA\_DST\_INC\_32 | UDMA\_ARB\_1);
341. /\* Set up the transfer parameters for the ADC0 Sequencer 2 primary control
342. \* structure. The mode is Basic mode so it will run to completion. \*/
343. MAP\_uDMAChannelTransferSet(UDMA\_CH16\_ADC0\_2 | UDMA\_PRI\_SELECT,
344. UDMA\_MODE\_PINGPONG,
345. (**void** \*)&ADC0->SSFIFO2, (**void** \*)&dstBufferA,
346. **sizeof**(dstBufferA)/4);
348. /\* Configure the control parameters for the alternate control structure for
349. \* the ADC0 Sequencer 2 channel. The alternate control structure is used for
350. \* copying the data from ADC0 Sequencer 2 FIFO to dstBufferB. The transfer
351. \* data size is 32 bits and the source address is not incremented while
352. \* the destination address is incremented at 32-bit boundary.
353. \*/
354. MAP\_uDMAChannelControlSet(UDMA\_CH16\_ADC0\_2 | UDMA\_ALT\_SELECT,
355. UDMA\_SIZE\_32 | UDMA\_SRC\_INC\_NONE |
356. UDMA\_DST\_INC\_32 | UDMA\_ARB\_1);
358. /\* Set up the transfer parameters for the ADC0 Sequencer 2 alternate
359. \* control structure. The mode is Basic mode so it will run to
360. \* completion \*/
361. MAP\_uDMAChannelTransferSet(UDMA\_CH16\_ADC0\_2 | UDMA\_ALT\_SELECT,
362. UDMA\_MODE\_PINGPONG,
363. (**void** \*)&ADC0->SSFIFO2, (**void** \*)&dstBufferB,
364. **sizeof**(dstBufferB)/4);
366. /\* The uDMA ADC0 Sequencer 2 channel is primed to start a transfer. As
367. \* soon as the channel is enabled and the Timer will issue an ADC trigger,
368. \* the ADC will perform the conversion and send a DMA Request. The data
369. \* transfers will begin. \*/
370. MAP\_uDMAChannelEnable(UDMA\_CH16\_ADC0\_2);
372. /\* Enable Timer-0 clock and configure the timer in periodic mode with
373. \* a frequency of 1 KHz. Enable the ADC trigger generation from the
374. \* timer-0. \*/
375. MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);
376. **while**(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_TIMER0)))
377. {
378. }

381. MAP\_TimerADCEventSet(TIMER0\_BASE, TIMER\_ADC\_TIMEOUT\_A);
382. timer\_fre\_set(SAMP\_FREQ,systemClock);
383. UARTprintf("code begin\r\n");
384. OLED\_ShowChar(40,1,'P',16);
385. /\* While loop to process the data \*/
386. **while**(1)
387. {
388. /\* Wait for Primary channel to complete and then clear the flag and
389. \* initialize the variables \*/
391. **while**((!setBufAReady)||(button\_value=((GPIOJ->DATA)&BIT0)));
392. **while**(!((GPIOJ->DATA)&BIT0));
393. setBufAReady=**false**;
394. old\_temp\_maxv=0;
396. //get\_window(WINDOW\_RECTANG,NUM\_SAMPLES, fftwin);
397. /\* First convert the sampled data to floating point format as the RMS
398. \* and DC average is being computed \*/
400. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
401. 第一步（用200KHz采样率精确测量基波频率）（PS：下述测量方法当FS较低时，测量低频会导致NaN出现
402. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
403. SAMP\_FREQ = 1024\*200;
404. timer\_fre\_set(SAMP\_FREQ,systemClock);
405. {GPIOK->DATA&=~BIT3;GPIOK->DATA&=~BIT2;}
406. i=5000000;
407. **while**(--i);
408. start\_adc\_convert();
410. **for**(step2\_vol\_index=0;step2\_vol\_index<NUM\_SAMPLES;step2\_vol\_index++)
411. {
412. step2\_vol\_list[step2\_vol\_index]=(**float**)(dstBufferA[step2\_vol\_index])\*3.3f/4096.0f;
413. }
414. arm\_max\_f32(step2\_vol\_list,NUM\_SAMPLES,&step2\_vinmax,&step2\_vinmax\_index);
415. arm\_min\_f32(step2\_vol\_list,NUM\_SAMPLES,&step2\_vinmin,&step2\_vinmin\_index);
416. step2\_vpp=step2\_vinmax-step2\_vinmin;
417. **if**(step2\_vpp<0.06)
418. {GPIOK->DATA|=BIT2;GPIOK->DATA&=~BIT3;}
419. **else** **if**(step2\_vpp<0.1)
420. {GPIOK->DATA|=BIT3;GPIOK->DATA&=~BIT2;}
421. **else** **if**(step2\_vpp<0.6)
422. {GPIOK->DATA&=~BIT3;GPIOK->DATA&=~BIT2;}
424. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
425. 第二步（根据基波频率调整采样率）
426. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
427. **do**
428. {
429. **while**(!setBufAReady);
430. setBufAReady=**false**;
431. SAMP\_FREQ = 1024\*200;
432. timer\_fre\_set(SAMP\_FREQ,systemClock);
433. start\_adc\_convert();
434. i=50;
435. **while**(--i);
436. **for**(step1\_circle\_i1=0;step1\_circle\_i1<5;step1\_circle\_i1++)
437. {
438. step1\_circle\_i2=2;
439. **while** (--step1\_circle\_i2)
440. {
441. **for**(t=0;t<NUM\_SAMPLES;t++)
442. {
443. //fft\_winout\_sampling[t]=fftwin[t]\*(float)dstBufferA[t];
444. fft\_winout\_sampling[t]=1\*(**float**)dstBufferA[t];
445. }
447. **for**(fft1\_i1=0;fft1\_i1<NUM\_SAMPLES;fft1\_i1++)
448. {
449. temp\_c\_buf[fft1\_i1\*2]=fft\_winout\_sampling[fft1\_i1]\*3.3f/4096.0f;
450. }

453. arm\_copy\_f32(temp\_c\_buf,temp\_in\_c\_buf,NUM\_SAMPLES\*2);

456. **for**(fft1\_i2=0;fft1\_i2<cir\_times;fft1\_i2++)
457. {
459. **for**(fft1\_i1=0;fft1\_i1<NUM\_SAMPLES;fft1\_i1++)
460. {
461. temp\_c\_buf[fft1\_i1\*2]=temp\_in\_c\_buf[fft1\_i1\*2]\*arm\_cos\_f32(-fft1\_i1\*fft1\_i2\*(2\*PI/(**float**)(NUM\_SAMPLES\*cir\_times)));
462. temp\_c\_buf[fft1\_i1\*2+1]=temp\_in\_c\_buf[fft1\_i1\*2]\*arm\_sin\_f32(-fft1\_i1\*fft1\_i2\*(2\*PI/(**float**)(NUM\_SAMPLES\*cir\_times)));
464. }
466. //fft
467. arm\_cfft\_f32(&arm\_cfft\_sR\_f32\_len1024,temp\_c\_buf,0,1);
468. arm\_cmplx\_mag\_f32(temp\_c\_buf,temp\_mag\_buf,NUM\_SAMPLES);

471. true\_fout[0]=(**float**)fft1\_i2\*((**float**)SAMP\_FREQ/((**float**)(NUM\_SAMPLES\*cir\_times)));
472. true\_vout[0]=temp\_mag\_buf[0]/(**float**)NUM\_SAMPLES;
473. fftout\_max\_info[3]=true\_vout[0];
475. **for**(fft1\_i1=1;fft1\_i1<NUM\_SAMPLES;fft1\_i1++)
476. {
477. true\_fout[fft1\_i1]=(**float**)fft1\_i1\*(**float**)SAMP\_FREQ/(**float**)NUM\_SAMPLES+fft1\_i2\*(**float**)SAMP\_FREQ/10240.0f;
479. true\_vout[fft1\_i1]=temp\_mag\_buf[fft1\_i1]/((**float**)NUM\_SAMPLES/2.0f);
480. }
482. **for**(fft1\_i1=0;fft1\_i1<3;fft1\_i1++)
483. {
484. true\_vout[fft1\_i1]=0;
486. }


490. arm\_max\_f32(true\_vout,NUM\_SAMPLES/2,&temp\_maxv,&temp\_maxv\_index);
491. **if**(temp\_maxv>old\_temp\_maxv)
492. {
493. old\_temp\_maxv=temp\_maxv;
494. fftout\_max\_info[1]=true\_fout[temp\_maxv\_index];
495. fftout\_max\_info[0]=temp\_maxv\_index;
496. arm\_copy\_f32(true\_vout,best\_vout\_buf,NUM\_SAMPLES);
497. arm\_copy\_f32(true\_fout,best\_fout\_buf,NUM\_SAMPLES);
499. }
501. }
502. fftout\_max\_info[2]=old\_temp\_maxv;

505. }
506. fft\_f\_out\_finnal\_value[step1\_circle\_i2]=fftout\_max\_info[1];
508. }
509. step1\_basefre=(fft\_f\_out\_finnal\_value[0]+fft\_f\_out\_finnal\_value[1]+fft\_f\_out\_finnal\_value[2]+fft\_f\_out\_finnal\_value[3]+fft\_f\_out\_finnal\_value[4])/5;
510. basefreq=fftout\_max\_info[1];
511. **if**(basefreq<=20000)
512. {
513. up\_max[0]=   fftout\_max\_info[2];
514. **for**(fft1\_i1=1;fft1\_i1<5;fft1\_i1++)
515. {
516. up\_max[fft1\_i1]=best\_vout\_buf[(**int**)fftout\_max\_info[0]\*(fft1\_i1+1)];
517. up\_max[fft1\_i1]/=up\_max[0];
518. }
519. up\_max[0]=1;
521. **if**(fftout\_max\_info[0]\*5<NUM\_SAMPLES/2)
522. thd=100.0f\*sqrt(up\_max[1]\*up\_max[1]+up\_max[2]\*up\_max[2]+up\_max[3]\*up\_max[3]+up\_max[4]\*up\_max[4])/(up\_max[0]);
523. **else**
524. thd=-1;
525. }
526. }**while**(thd>=50||fftout\_max\_info[1]<1000);

529. /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*
530. \* 第三步（带通采样）
531. \*
532. \*
533. \*
534. \*
535. \*
536. \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/
537. **if**(fftout\_max\_info[1]>20000)
538. {
539. low\_freq=calculate\_FS(fftout\_max\_info[1]);
540. SAMP\_FREQ = low\_freq;
541. timer\_fre\_set(SAMP\_FREQ,systemClock);
542. **do**
543. {
545. start\_adc\_convert();
546. i=50;
547. **while**(--i);
548. **for**(t=0;t<NUM\_SAMPLES;t++)
549. {
550. //fft\_winout\_sampling[t]=fftwin[t]\*(float)dstBufferA[t];
551. temp\_c\_buf[t\*2]=1\*(**float**)dstBufferA[t];
552. }
553. arm\_cfft\_f32(&arm\_cfft\_sR\_f32\_len1024,temp\_c\_buf,0,1);
554. arm\_cmplx\_mag\_f32(temp\_c\_buf,temp\_mag\_buf,NUM\_SAMPLES);
556. true\_fout[0]=0;
557. true\_vout[0]=temp\_mag\_buf[0]/(**float**)NUM\_SAMPLES;
558. fftout\_max\_info[3]=true\_vout[0];
560. **for**(fft1\_i1=1;fft1\_i1<NUM\_SAMPLES;fft1\_i1++)
561. {
562. true\_fout[fft1\_i1]=(**float**)fft1\_i1\*(**float**)SAMP\_FREQ/(**float**)NUM\_SAMPLES;
564. true\_vout[fft1\_i1]=temp\_mag\_buf[fft1\_i1]/((**float**)NUM\_SAMPLES/2.0f);
565. }
567. **for**(fft1\_i1=0;fft1\_i1<3;fft1\_i1++)
568. {
569. true\_vout[fft1\_i1]=0;
571. }
573. arm\_max\_f32(true\_vout,NUM\_SAMPLES/2,&temp\_maxv,&temp\_maxv\_index);
574. up\_max[0]=temp\_maxv;
575. max\_index[0]=temp\_maxv\_index;
576. **for**(fft1\_i1=1;fft1\_i1<5;fft1\_i1++)
577. {
578. max\_index[fft1\_i1]=(fft1\_i1+1)\*max\_index[0];
579. **if**(max\_index[fft1\_i1]>512)
580. max\_index[fft1\_i1]=1024-max\_index[fft1\_i1];
581. **if**(max\_index[fft1\_i1]<0)
582. max\_index[fft1\_i1]=-max\_index[fft1\_i1];
583. **if**(max\_index[fft1\_i1]>512)
584. max\_index[fft1\_i1]=1024-max\_index[fft1\_i1];
585. **if**(max\_index[fft1\_i1]<0)
586. max\_index[fft1\_i1]=-max\_index[fft1\_i1];
587. up\_max[fft1\_i1]=true\_vout[max\_index[fft1\_i1]]/up\_max[0];
588. }
589. up\_max[0]=1;
590. thd=100.0f\*sqrt(up\_max[1]\*up\_max[1]+up\_max[2]\*up\_max[2]+up\_max[3]\*up\_max[3]+up\_max[4]\*up\_max[4])/(up\_max[0]);
591. }**while**(thd>=50.0f);

594. }
596. **if**(thd>34.5&&thd<37) thd+=1.5;
598. UARTStdioConfig(0, 115200, systemClock);
599. /\* 发送THD \*/
600. UARTprintf("x0.val=%d", (**int**)(thd\*100000.0f));
601. UARTprintf("%s", screen\_end);
602. UARTprintf("x0.vvs1=5");
603. UARTprintf("%s", screen\_end);
605. /\* 发送归一化幅值\*/
606. UARTprintf("x1.val=%d", (**int**)(up\_max[0]\*100000));
607. UARTprintf("%s", screen\_end);
608. UARTprintf("x1.vvs1=5");
609. UARTprintf("%s", screen\_end);
611. UARTprintf("x2.val=%d", (**int**)(up\_max[1]\*100000));
612. UARTprintf("%s", screen\_end);
613. UARTprintf("x2.vvs1=5");
614. UARTprintf("%s", screen\_end);
616. UARTprintf("x3.val=%d", (**int**)(up\_max[2]\*100000));
617. UARTprintf("%s", screen\_end);
618. UARTprintf("x3.vvs1=5");
619. UARTprintf("%s", screen\_end);
621. UARTprintf("x4.val=%d", (**int**)(up\_max[3]\*100000));
622. UARTprintf("%s", screen\_end);
623. UARTprintf("x4.vvs1=5");
624. UARTprintf("%s", screen\_end);
626. UARTprintf("x5.val=%d", (**int**)(up\_max[4]\*100000));
627. UARTprintf("%s", screen\_end);
628. UARTprintf("x5.vvs1=5");
629. UARTprintf("%s", screen\_end);
631. /\* 绘制波形曲线 \*/
632. **for**(wave\_count = 0; wave\_count < 250; wave\_count++){
633. wave[wave\_count] = up\_max[0]\*sin(wave\_count\*PI/125)   +
634. up\_max[1]\*sin(2\*wave\_count\*PI/125) +
635. up\_max[2]\*sin(3\*wave\_count\*PI/125) +
636. up\_max[3]\*sin(4\*wave\_count\*PI/125) +
637. up\_max[4]\*sin(5\*wave\_count\*PI/125);
638. }
640. **for**(wave\_count = 0; wave\_count < 250; wave\_count++){
641. UARTprintf("add s0.id,0,%d\xff\xff\xff",(**int**)(wave[wave\_count]\*50)+100);
642. }
643. //            UARTprintf("%d", (int)(wave[249]/2\*255) + 120);
645. UARTStdioConfig(2, 115200, systemClock);
646. UARTprintf("PKT %d %d %d %d %d %d 200000",(**int**)(thd\*100000.0f),(**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));
648. }
649. }

### 附件4 ESP32部分代码

1. #include <stdio.h>
2. #include <string.h>
3. #include "sdkconfig.h"
4. #include "driver/gpio.h"
5. #include "lwip/err.h"
6. #include "lwip/sys.h"
7. #include "esp\_err.h"
8. #include "esp\_log.h"
9. #include "esp\_wifi.h"
10. #include "esp\_event.h"
11. #include "esp\_system.h"
12. #include "esp\_spi\_flash.h"
13. #include "nvs\_flash.h"
14. #include "driver/uart.h"
15. #include "lwip/sockets.h"
16. #include "freertos/task.h"
17. #include "freertos/FreeRTOS.h"
19. **static** **const** **char**\* TAG = "MyModule";
21. #define TCP\_SERVER\_PORT 8888            // TCP服务器端口号
23. #define TXD2\_PIN        GPIO\_NUM\_17     // TXD2引脚号
24. #define RXD2\_PIN        GPIO\_NUM\_16     // RXD2引脚号
26. #define BUFF\_SIZE       2048            // 缓冲区大小
28. **int** client\_socket = 0;
29. uint8\_t buff[BUFF\_SIZE];

32. **static** esp\_err\_t event\_handler(**void**\* ctx, system\_event\_t\* event){
33. **switch**(event->event\_id){
34. **case** SYSTEM\_EVENT\_WIFI\_READY:
35. printf("[Wifi]......Ready\n");
36. **break**;
38. **case** SYSTEM\_EVENT\_AP\_START:
39. printf("[Wifi]......Start\n");
40. **break**;
42. **case** SYSTEM\_EVENT\_AP\_STACONNECTED:
43. printf("[Wifi]......Connected\n");
44. **break**;
46. **case** SYSTEM\_EVENT\_AP\_STADISCONNECTED:
47. printf("[Wifi]......Disconnected\n");
48. **break**;
50. **default**:
51. **break**;
52. }
54. **return** ESP\_OK;
55. }
57. **void** wifi\_init(){
58. //初始化tcpip
59. tcpip\_adapter\_init();
61. //设置回调函数为event\_handler
62. ESP\_ERROR\_CHECK(esp\_event\_loop\_init(event\_handler, NULL));
64. //初始化wifi设置
65. wifi\_init\_config\_t cfg = WIFI\_INIT\_CONFIG\_DEFAULT();
66. ESP\_ERROR\_CHECK(esp\_wifi\_init(&cfg));
68. //配置wifi
69. wifi\_config\_t wifi\_config = {
70. .ap = {
71. .ssid           =   "CCC",                      //WIFI名称
72. .ssid\_len       =   3,                          //WIFI名长度
73. .max\_connection =   1,                          //最大连接数
74. .password       =   "12345678",                 //WIFI密码
75. .authmode       =   WIFI\_AUTH\_WPA\_WPA2\_PSK,     //WIFI加密方式
76. },
77. };
78. ESP\_ERROR\_CHECK(esp\_wifi\_set\_mode(WIFI\_MODE\_AP));
79. ESP\_ERROR\_CHECK(esp\_wifi\_set\_config(ESP\_IF\_WIFI\_AP, &wifi\_config));
81. //启动WIFI
82. ESP\_ERROR\_CHECK(esp\_wifi\_start());
83. }
85. **void** uart\_init(){
86. /\* uart配置结构体 \*/
87. uart\_config\_t uart2\_config = {
88. .baud\_rate = 115200,                     // 波特率
89. .data\_bits = UART\_DATA\_8\_BITS,           // 数据位
90. .parity    = UART\_PARITY\_DISABLE,        // 校验位
91. .stop\_bits = UART\_STOP\_BITS\_1,           // 停止位
92. .flow\_ctrl = UART\_HW\_FLOWCTRL\_DISABLE    // 硬件流控
93. };
94. uart\_param\_config(UART\_NUM\_2, &uart2\_config);
96. /\* IO端口映射 \*/
97. uart\_set\_pin(UART\_NUM\_2, TXD2\_PIN, RXD2\_PIN, UART\_PIN\_NO\_CHANGE, UART\_PIN\_NO\_CHANGE);
99. /\* 使能串口并设置缓冲区大小 \*/
100. uart\_driver\_install(UART\_NUM\_2, BUFF\_SIZE\*2, BUFF\_SIZE\*2, 0, NULL, 0);
101. }
103. **void** tcp\_server\_init(**int**\* client\_socket){
104. //1.新建socket
105. **int** server\_socket = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_IP);
106. **if**(server\_socket < 0){
107. printf("[TCP]......Create socket failed\n");
108. }
110. //2.配置服务器
111. **struct** sockaddr\_in dest\_addr = {
112. .sin\_family         =   AF\_INET,
113. .sin\_addr.s\_addr    =   htonl(INADDR\_ANY),
114. .sin\_port           =   htons(TCP\_SERVER\_PORT),
115. };
117. //3.绑定地址
118. **int** err = bind(server\_socket, (**struct** socketassr\*)&dest\_addr, **sizeof**(dest\_addr));
119. **if**(err){
120. printf("[TCP]......Bind failed\n");
121. close(server\_socket);
122. }
124. //4.开始监听
125. err = listen(server\_socket, 5);
126. **if**(err){
127. printf("[TCP]......Listen failed\n");
128. close(server\_socket);
129. }
131. //5.等待客户端连接
132. **while**(1){
133. **struct** sockaddr\_in6 source\_addr;
134. uint32\_t addr\_len = **sizeof**(source\_addr);
135. \*(client\_socket) = accept(server\_socket, (**struct** sockaddr\*)&source\_addr, &addr\_len);
136. **if**(\*(client\_socket) < 0){
137. printf("[TCP]......Connect failed\n");
138. close(server\_socket);
139. }
141. **else**{
142. printf("[TCP]......Connected\n");
143. **break**;
144. }
145. }
146. }
148. **void** uart2\_rx\_task(){
149. **while**(1){
150. memset(buff, 0, **sizeof**(buff));
151. **const** **int** len = uart\_read\_bytes(UART\_NUM\_2, buff, BUFF\_SIZE, 300/portTICK\_RATE\_MS);
153. **if**(len > 0){
154. send(client\_socket, buff, len, 0);
155. }
157. vTaskDelay(100/portTICK\_RATE\_MS);
158. }
159. }

162. **void** app\_main(**void**){
164. /\* 初始化NVS \*/
165. ESP\_ERROR\_CHECK(nvs\_flash\_init());
167. /\* 初始化WIFI和UART \*/
168. wifi\_init();
169. uart\_init();
171. /\* 初始化TCP \*/
172. tcp\_server\_init(&client\_socket);
174. /\* 创建uart2串口任务 \*/
175. xTaskCreate(uart2\_rx\_task, "uart2\_rx\_task", BUFF\_SIZE\*2, NULL, configMAX\_PRIORITIES, NULL);
176. ESP\_LOGI(TAG, "Create task done!\n");
177. }

### 附件5 安卓上位机部分代码

1. package com.example.viewdemo;
3. import androidx.annotation.NonNull;
4. import androidx.appcompat.app.AppCompatActivity;
6. import android.content.Intent;
7. import android.graphics.Canvas;
8. import android.os.Bundle;
9. import android.os.Handler;
10. import android.os.Message;
11. import android.renderscript.ScriptGroup;
12. import android.util.Log;
13. import android.view.View;
14. import android.widget.Button;
15. import android.widget.TextView;
16. import android.widget.Toast;
18. import java.io.IOException;
19. import java.io.InputStream;
20. import java.io.OutputStream;
21. import java.net.Socket;
22. import java.nio.charset.StandardCharsets;
23. import java.util.ArrayList;
24. import java.util.Random;
26. /\*
27. \* Target: 安卓上位机，可与服务器建立TCP连接，显示数据波形
28. \*         软件特性：
29. \*                 1.可以自己设置服务器IP和端口号
30. \*                 2.波形显示为Auto模式
31. \*         通信协议：
32. \*                 1.发送数据给上位机前需要先发送 "READY"
33. \*                 2.数据包由207个数据组成，数据之间用空格隔开
34. \*                 3.数据包具体内容为"PKT THD U0 U1 U2 U3 U4 U5 Vpp point1 point2 ... point200"
35. \* Date: 2021/5/8 23:26
36. \* \*/
37. **public** **class** MainActivity extends AppCompatActivity {
39. **public** **static** **float** PI = (**float**) 3.1415926;
41. /\*\*默认IP地址和端口号\*/
42. **public** **static** String IP\_address = null;
43. **public** **static** **int** St\_address = 0;
45. /\*\*Handler状态\*/
46. **public** **static** final **int** REFRESH = 1;
47. **public** **static** final **int** CONNECTED = 0;
49. /\*\*要绘制的数据\*/
50. **public** **static** **float**[] floats = **new** **float**[200];
52. /\*\*频率、有效值、峰峰值\*/
53. **public** **static** String thd, frq, u1, u2, u3, u4, u5, pp;
55. /\*\*缓冲区大小，单位: 字节\*/
56. **public** **static** final **int** BUFF\_SIZE = 2500;
58. /\*\*日志筛选\*/
59. **private** **static** final String TAG = "MainActivity";
61. @Override
62. **protected** **void** onCreate(Bundle savedInstanceState) {
63. super.onCreate(savedInstanceState);
64. setContentView(R.layout.activity\_main);
66. /\*\*初始化图像\*/
67. DrawLineChart chart = findViewById(R.id.chart);
68. chart.setMaxValue(10f);
69. chart.setMinValue(-10f);
70. chart.setNumberLine(5);
72. /\*\*初始化数据\*/
73. **for** (**int** i = 0; i < floats.length; i++) {
74. floats[i] = 0;
75. }
76. chart.setValue(floats);
77. getSupportActionBar().hide();
79. /\*\*跳转到设置界面\*/
80. Button set = (Button) findViewById(R.id.set);
81. set.setOnClickListener(**new** View.OnClickListener() {
82. @Override
83. **public** **void** onClick(View v) {
84. Intent intent  = **new** Intent(MainActivity.**this**, Connect.**class**);
85. startActivityForResult(intent, 1);
86. }
87. });
89. /\*\*点击进行连接和数据传输\*/
90. Button connect = (Button) findViewById(R.id.connect);
91. connect.setOnClickListener(**new** View.OnClickListener() {
92. @Override
93. **public** **void** onClick(View v) {
95. /\*\*若未填写信息则弹出警告\*/
96. **if**(IP\_address == null || St\_address == 0){
97. Toast.makeText(MainActivity.**this**, "请先点击设置", Toast.LENGTH\_SHORT).show();
98. }
100. **else** {
101. **new** ConnectThread().start();
102. }
103. }
104. });
105. }

108. /\*\*创建Handler用于更新曲线\*/
109. **private** Handler handler = **new** Handler(){
110. @Override
111. **public** **void** handleMessage(Message msg) {
112. **if**(msg.what == REFRESH){
113. /\*\*更新曲线\*/
114. DrawLineChart chart = findViewById(R.id.chart);
116. chart.setMinValue((-1) \* getNum(pp));
117. chart.setMaxValue(getNum(pp));
118. chart.setNumberLine(5);
119. chart.setValue(floats);
120. chart.requestLayout();
122. /\*\*更新THD\*/
123. TextView THD = (TextView)findViewById(R.id.THD);
124. THD.setText("THD: " + thd + "%");
126. /\*\*更新频率\*/
127. TextView FRQ = (TextView)findViewById(R.id.FRQ);
128. FRQ.setText("Frq: " + frq + "Hz");
130. /\*\*更新归一化幅值\*/
131. TextView U1 = (TextView)findViewById(R.id.U1);
132. U1.setText("U1: " + u1);
134. TextView U2 = (TextView)findViewById(R.id.U2);
135. U2.setText("U2: " + u2);
137. TextView U3 = (TextView)findViewById(R.id.U3);
138. U3.setText("U3: " + u3);
140. TextView U4 = (TextView)findViewById(R.id.U4);
141. U4.setText("U4: " + u4);
143. TextView U5 = (TextView)findViewById(R.id.U5);
144. U5.setText("U5: " + u5);
145. }
147. **else** **if**(msg.what == CONNECTED){
148. Toast.makeText(MainActivity.**this**, "已连接", Toast.LENGTH\_SHORT).show();
149. }
150. }
151. };

154. /\*\*连接线程\*/
155. **private** **class** ConnectThread extends Thread{
156. @Override
157. **public** **void** run() {
158. Message msg = **new** Message();
160. **try** {
161. Socket s = **new** Socket(IP\_address, St\_address);
162. InputStream is = s.getInputStream();
163. OutputStream os = s.getOutputStream();
164. byte[] b = **new** byte[BUFF\_SIZE];
166. /\*\*成功连接\*/
167. msg.what = CONNECTED;
168. handler.sendMessage(msg);
170. /\*\*发送ID码验证身份\*/
171. //os.write("Phone".getBytes());
173. /\*\*线程阻塞\*/
174. **while** (**true**){
175. // 读取信息
176. **int** len = is.read(b);
177. String message = **new** String(b, 0, len);
178. Log.d(TAG, "Received Data:" + message);


182. // 分割数据流
183. String[] data = message.split(" ");
184. Log.d(TAG, "DataLength:" + data.length);
186. **if**(data[0].equals("PKT") && data.length == 9){
187. // 提取THD、归一化幅值
188. thd = String.valueOf(Float.parseFloat(data[1])/1000);
189. frq = data[2]+"00";
190. Log.i(TAG, "FRQ:" + frq);
191. u1 = String.valueOf(Float.parseFloat(data[3]));
192. u2 = String.valueOf(Float.parseFloat(data[4])/100000);
193. u3 = String.valueOf(Float.parseFloat(data[5])/100000);
194. u4 = String.valueOf(Float.parseFloat(data[6])/100000);
195. u5 = String.valueOf(Float.parseFloat(data[7])/100000);
196. pp = String.valueOf(Float.parseFloat(data[8]));
198. // 提取波形信息
199. **for**(**int** i = 0; i < 200; i++){
200. floats[i] = (**float**) (Float.parseFloat(u1)\*(Math.sin(i\*PI/100)))    +
201. (**float**) (Float.parseFloat(u2)\*(Math.sin(2\*i\*PI/100)))  +
202. (**float**) (Float.parseFloat(u3)\*(Math.sin(3\*i\*PI/100)))  +
203. (**float**) (Float.parseFloat(u4)\*(Math.sin(4\*i\*PI/100)))  +
204. (**float**) (Float.parseFloat(u5)\*(Math.sin(5\*i\*PI/100)));
205. floats[i] = floats[i];
206. }
208. /\*\*更新UI\*/
209. Message msg0 = **new** Message();
210. msg0.what = REFRESH;
211. handler.sendMessage(msg0);
212. }
213. }
214. } **catch** (IOException e) {
215. e.printStackTrace();
216. }
217. }
218. }
220. /\*\*接受设置界面传递的参数\*/
221. @Override
222. **protected** **void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {
224. **if**(data != null){
225. super.onActivityResult(requestCode, resultCode, data);
227. /\*\*获取IP地址和端口号\*/
228. **if** (requestCode == 1) {
229. IP\_address = data.getStringExtra("IP\_address");
230. St\_address = data.getIntExtra("St\_address", 0);
231. }
232. }
234. }
236. /\*\*将字符串中的数字提取出来\*/
237. **public** **float** getNum(String str){
238. **int** index = str.length();
240. **for**(**int** i = 0; i < str.length(); i++){
241. **if**(str.charAt(i) > '9'){
242. index = i;
243. **break**;
244. }
245. }
247. **return** Float.parseFloat(str.substring(0, index));
248. }
249. }