FD0042

**2021年全国大学生电子设计竞赛**

**信号失真度测量装置（A题）**

【本科组】



2021年11月7日

**摘 要**

本设计是一信号失真度测量装置，可对来自函数或任意波形发生器的周期信号进行采集分析，测得输入信号的总谐波失真 THD，并可在手机上显示测量信息。

系统由处理器模块、信号输入模块、数据显示模块、移动设备通信模块组成。处理器模块采用TI的MSP432E401Y开发板。信号输入模块对输入信号添加直流偏置并放大后传输给处理器模块，进行AD转换和FFT变换， 并将处理结果（失真度、波形等）发送给数据显示模块显示，同时可以用手机连接蓝牙传输模块，在对应APP中显示同上内容。

经测试，本设计性能指标部分达到设计要求。

目 录

[一、系统方案 1](#_Toc87200307)

[1.1 处理器模块的分析与选择 1](#_Toc87200308)

[1.2 信号输入电路的分析与选择 1](#_Toc87200309)

[1.3 数据处理方案的分析与选择 1](#_Toc87200310)

[1.4 移动设备通信模块的分析与选择 2](#_Toc87200311)

[二、理论分析与计算 2](#_Toc87200312)

[2.1 直流偏置与信号放大电路 2](#_Toc87200313)

[2.2 FFT（快速傅里叶变换） 2](#_Toc87200314)

[2.3 THD（总谐波失真） 2](#_Toc87200315)

[三、电路与程序设计 3](#_Toc87200316)

[3.1 电路的设计 3](#_Toc87200317)

[3.1.1 系统总体框图 3](#_Toc87200318)

[3.1.2 信号输入模块 3](#_Toc87200319)

[3.2 程序的设计 4](#_Toc87200320)

[3.2.1 程序功能描述与设计思路 4](#_Toc87200321)

[3.2.2 程序流程图 5](#_Toc87200322)

[四、测试方案与测试结果 5](#_Toc87200323)

[4.1 测试方案 5](#_Toc87200324)

[4.1.1 软件仿真 5](#_Toc87200325)

[4.1.2 硬件测试 6](#_Toc87200326)

[4.2 测试结果及分析 6](#_Toc87200327)

[4.2.1 测试结果 6](#_Toc87200328)

[4.2.2 测试分析与结论 6](#_Toc87200329)

[附录1：电路原理图 7](#_Toc87200330)

[附录2：源程序 7](#_Toc87200331)

[附录3：参考文献 16](#_Toc87200332)

一、系统方案

本次设计的主要思路为第一部分对信号进行处理，实现信号的抬升和基本的放大倍数，使得信号进入AD采样范围。AD采样后采用MSP432E401Y开发板进行数据处理，采用FFT计算库进行数据处理，从而达到基本性能指标的测试。最后通过OLED进行间隔2S的三页分别显示。本设计主要采用TI的MSP432E401Y开发板、模拟信号调理电路、数据显示模块、移动设备通信模块几个主要组成部分。

1.1 处理器模块的分析与选择

根据题目要求，本题用于信号失真度测量的主控制器和数据采集器使用 TI 公司的MCU及其片内 ADC，经过对性能需求的综合分析，选择MSP432E401Y开发板作为处理器模块。

MSP432E401Y采用了120MHz Arm Cortex-M4F CPU、1MB 闪存、256kB SRAM 和先进的加密加速器，有 2个基于12位SAR的ADC模块，每个模块支持高达2Msps的采样率，集成了以太网 MAC 和一系列有线通信接口（包括 USB-OTG、CAN、Quad-SPI (QSSI)、I2C、SPI、UART 以及其他串行协议），完全可以满足本题对数据采集与处理的要求，且有一定的扩展空间。

1.2 信号输入电路的分析与选择

输入信号的处理方案我们采用加法器对小信号进行放大并且提升其直流偏置电压，从而达到AD采样的电压范围，由于时间有限，所以采用了平常较为常用的这种方法。这种方法的弊端是会造成噪声信号与输入信号同时放大，从而使得噪声对后续信号处理的影响更大，因此更优的方案应为采用差分输入，但是由于平时接触较少，因而并未选择此方案。

1.3 数据处理方案的分析与选择

数据处理的核心为FFT的变换，以及相关的频率和失真度的计算。但是需要关注在采样过程中不能采用运算和显示同步进行的情况，因为这样会导致前面的FFT变换我们采用了官方所给的库进行计算，而采集后的数据可以根据相应的计算公式进行算法的设计，从而对相关指标参数进行求解。

1.4 移动设备通信模块的分析与选择

与移动设备的通信采用ESP32。ESP32 集成了天线开关、射频 balun、功率放大器、低噪放大器、过滤器和电源管理模块，具有2.4 GHz Wi-Fi 加蓝牙双模芯片采用 TSMC 低功耗 40nm 技术，功耗性能和射频性能最佳，安全可靠，易于扩展至各种应用。

关于具体模式的选择，由于蓝牙功耗较低，并且针对一对一连接最优化，较为符合本设计的要求，故选择蓝牙模块进行数据传输。手机上采用可以从蓝牙串口进行数据通信的APP作为上位机，对数据进行实时接收和处理。

二、理论分析与计算

2.1 直流偏置与信号放大电路

前端信号最大为600mv所以需要进行电压的抬升并且进行放大，采用NE5523起到加法器和放大器的作用，使得电压在规定范围内进行AD采样，从而进行合理的转换。

2.2 FFT（快速傅里叶变换）

FFT的基本思想是把原始的N点序列，依次分解成一系列的短序列。充分利用DFT计算式中指数因子 所具有的对称性质和周期性质，进而求出这些短序列相应的DFT并进行适当组合，达到删除重复计算，减少乘法运算和简化结构的目的。此后，在这思想基础上又开发了高基和分裂基等快速算法，随着数字技术的高速发展，1976年出现建立在数论和多项式理论基础上的维诺格勒傅里叶变换算法(WFTA）和素因子傅里叶变换算法。它们的共同特点是，当N是素数时，可以将DFT算转化为求循环卷积，从而更进一步减少乘法次数，提高速度。

本题采用FFT算法对输入信号进行处理，得到相关数据。

2.3 THD（总谐波失真）

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, 出现谐波失真的放大器输出交流电压为, 则的总谐波失真（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

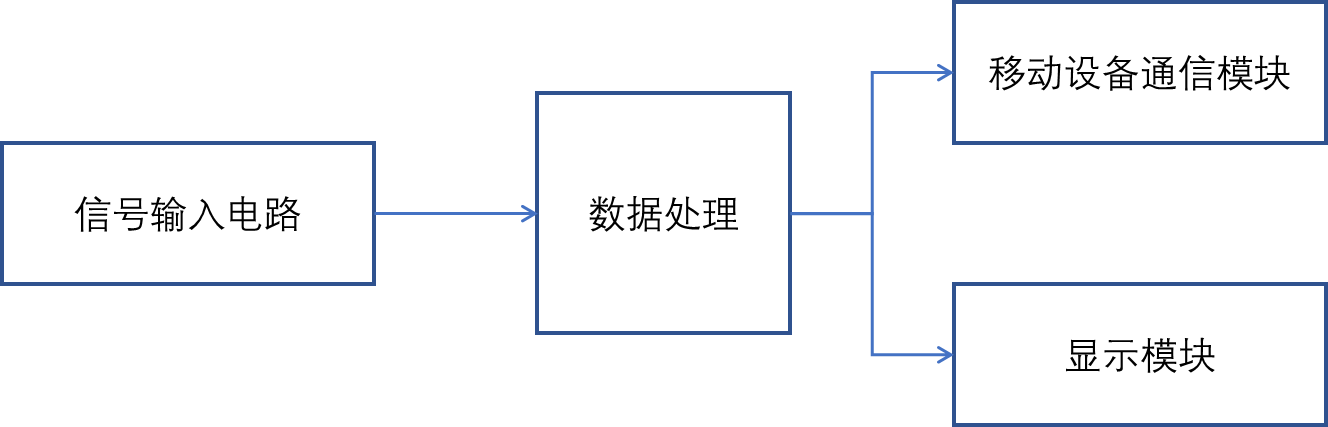
为本题失真度的标称值。

一般说来，1kHz频率处的总谐波失真最小，因此不少产品均以该频率的失真作为它的指标，本题的基础部分测试频率也为1kHz。

三、电路与程序设计

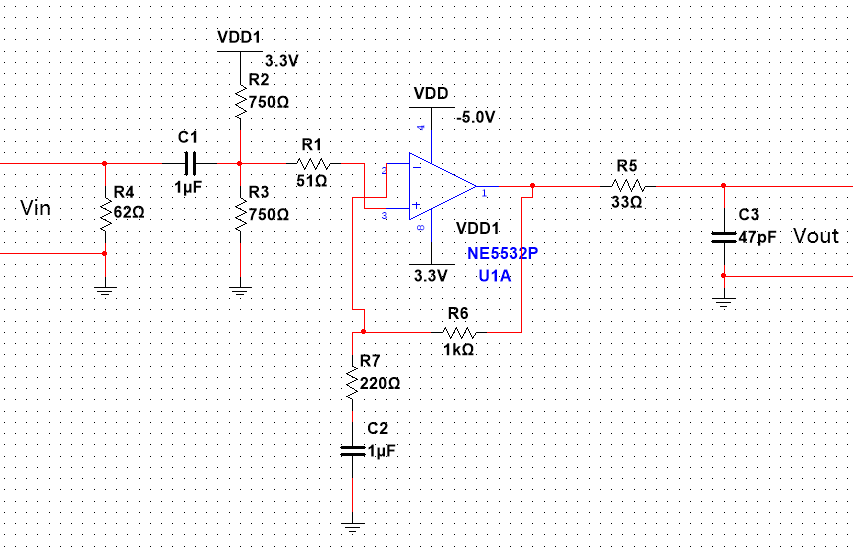
3.1 电路的设计

3.1.1 系统总体框图



3.1.2 信号输入模块

采用NE5523构成基本的加法器和抬高电压。



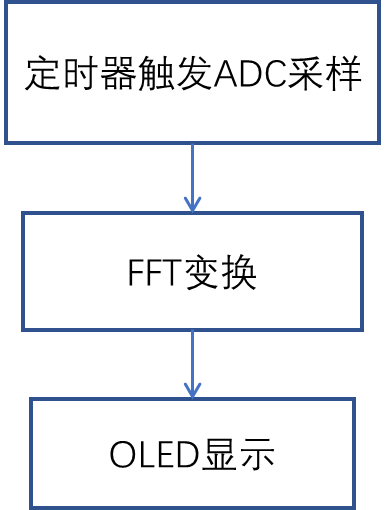
3.2 程序的设计

3.2.1 程序功能描述与设计思路

本次设计采用MSP432E的内置ADC，首先参考，官方代码模板，通过定时器触发ADC进行采样，根据题目要求采用TI的ADC和FFT转换，定时器时间到时，ADC进行采样，同时采用采样、转换、采样、转换的串行流程，设计代码，当每一组ADC采样为一组(4096个点)后，进行FFT转换，然后通过OLED进行三页显示，每页2秒，在10秒内完成题目内的要求通过OLED完成各种显示。

程序设计采用中断和顺序执行的经典思路，但是最重要的是要考虑到在采样过后的OLED输出显示中需要有一段时间的空闲时间，如何调整采样的时间、FFT转换时间、OLED显示时间是本次程序设计的关键所在。

3.2.2 程序流程图



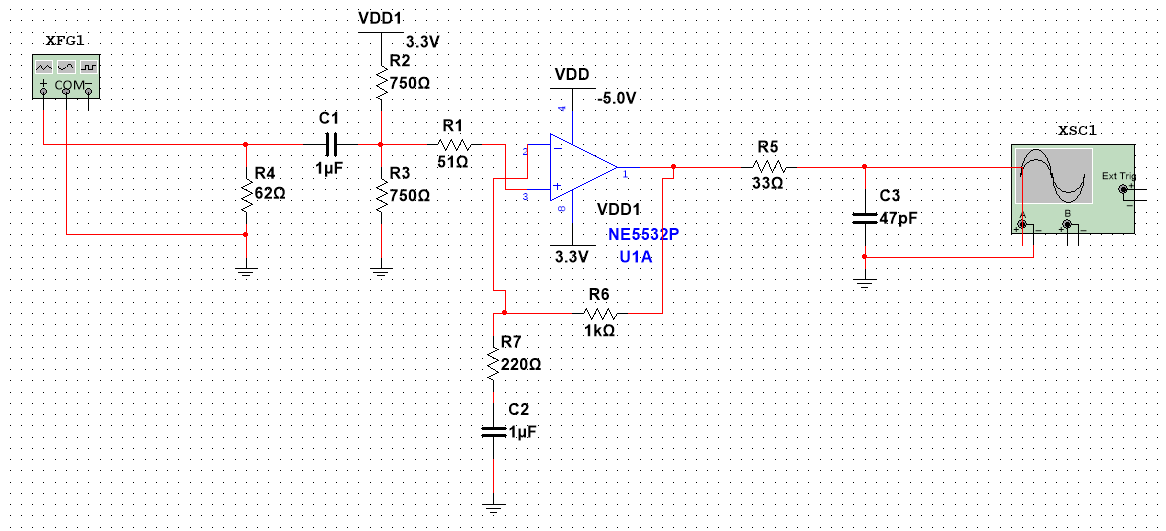
四、测试方案与测试结果

4.1 测试方案

4.1.1 软件仿真

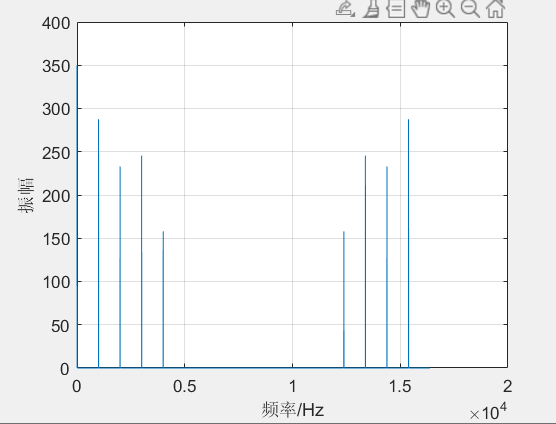
信号输入电路部分

采用软件仿真再实际测试的方法。使用Multisim软件对电路进行仿真。



数据处理部分

（1）使用Matlab软件进行FFT的仿真；



（2）使用Vofa+软件接收串口数据，验证输出。

4.1.2 硬件测试

将电路全部连接好，用函数发生器输出的周期信号作为测量装置的输入信号，调节不同频率、幅度和失真度，对所得失真度与理论失真度进行比较，计算差值。

4.2 测试结果及分析

4.2.1 测试结果

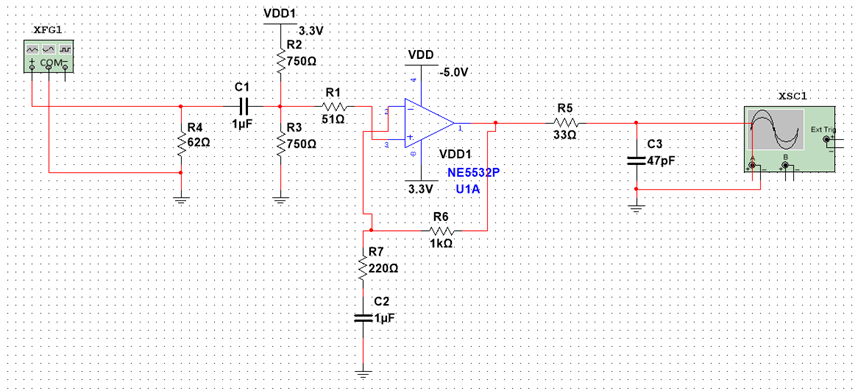
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 频率（kHz） | 幅度（mV） | THDo | THDx | Δ |
| 1 | 600 | 38% | 37% | 1% |
| 2 | 600 | 38% | 45% | 7% |
| 1 | 300 | 38% | 38% | 0% |
| 2 | 300 | 38% | 49% | 11% |

4.2.2 测试分析与结论

软件测试结果与计算一致，符合要求。

硬件测试结果在1kHz时与计算结果一致，一定程度上完成了设计的基本要求；检测频率提高后有较明显的误差，发挥部分没有完成。

附录1：电路原理图



附录2：源程序

bool setBufDisplayReady = false;

bool setClearDisplayReady = false;

int current\_tick = 15;

int current\_sec = 0 ;

int current\_page = 0;

/\* Global variables and defines for FFT \*/

#define NUM\_SAMPLES 4096

#define SAMP\_FREQ 16384

#define IFFTFLAG 0

#define BITREVERSE 1

volatile int16\_t fftOutput[NUM\_SAMPLES\*2];

/\* Global variables and defines for ADC \*/

static uint32\_t dstBufferProcess[NUM\_SAMPLES];

static uint32\_t dstBufferNoneProcess[NUM\_SAMPLES];

bool setBufProcessReady = false;

int transferIndex = 0;

uint32\_t getADCValue[1];

//volatile bool bgetConvStatus = false;;

/\* Global variables and defines for dcRMS \*/

volatile float32\_t rmsBuff;

volatile float32\_t dcBuff;

float32\_t rmsCalculation;

void ADC0SS2\_IRQHandler(void)

{

uint32\_t getIntStatus;

/\* Get the interrupt status from the ADC \*/

getIntStatus = MAP\_ADCIntStatus(ADC0\_BASE, 2, true);

/\* If the interrupt status for Sequencer-2 is set the

\* clear the status and read the data \*/

if(getIntStatus == 0x4)

{

/\* Clear the ADC interrupt flag. \*/

MAP\_ADCIntClear(ADC0\_BASE, 2);

/\* Read ADC Value. \*/

MAP\_ADCSequenceDataGet(ADC0\_BASE, 2, getADCValue);

/\* Self Jugle Part Start\*/

if(setBufProcessReady == false)

{

if(transferIndex<NUM\_SAMPLES)

{

dstBufferNoneProcess[transferIndex] = getADCValue[0] ;

dstBufferProcess[transferIndex] = getADCValue[0]\*7 ;

transferIndex ++ ;

}

else

{

setBufProcessReady = true ;

transferIndex = 0 ;

}

}

/\* Self Jugle Part End\*/

//bgetConvStatus = true;

}

}

void ConfigureUART(uint32\_t systemClock)

{

/\* Enable the clock to GPIO port A and UART 0 \*/

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOA);

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_UART0);

/\* Configure the GPIO Port A for UART 0 \*/

MAP\_GPIOPinConfigure(GPIO\_PA0\_U0RX);

MAP\_GPIOPinConfigure(GPIO\_PA1\_U0TX);

MAP\_GPIOPinTypeUART(GPIO\_PORTA\_BASE, GPIO\_PIN\_0 | GPIO\_PIN\_1);

/\* Configure the UART for 115200 bps 8-N-1 format \*/

UARTStdioConfig(0, 115200, systemClock);

}

int main(void)

{

uint32\_t systemClock;

//uint32\_t setFFTmaxValue;

//uint32\_t setFFTmaxFreqIndex;

//int\_fast32\_t i32IPart[3];

//int\_fast32\_t i32FPart[3];

/\* Configure the system clock for 120 MHz \*/

systemClock = MAP\_SysCtlClockFreqSet((SYSCTL\_XTAL\_25MHZ | SYSCTL\_OSC\_MAIN |

SYSCTL\_USE\_PLL | SYSCTL\_CFG\_VCO\_480),

120000000);

/\* Initialize serial console \*/

ConfigureUART(systemClock);

OLED\_Init();

OLED\_Clear();

/\* Enable the clock to GPIO Port E and wait for it to be ready \*/

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_GPIOE);

while(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_GPIOE)))

{

}

/\* Configure PE0-PE3 as ADC input channel \*/

MAP\_GPIOPinTypeADC(GPIO\_PORTE\_BASE, GPIO\_PIN\_1);

/\* Enable the clock to ADC-0 and wait for it to be ready \*/

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_ADC0);

while(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_ADC0)))

{

}

MAP\_ADCSequenceStepConfigure(ADC0\_BASE, 2, 0, ADC\_CTL\_CH2 | ADC\_CTL\_IE |

ADC\_CTL\_END);

/\* Enable sample sequence 2 with a timer signal trigger. Sequencer 2

\* will do a single sample when the timer generates a trigger on timeout\*/

MAP\_ADCSequenceConfigure(ADC0\_BASE, 2, ADC\_TRIGGER\_TIMER, 2);

/\* Since sample sequence 2 is now configured, it must be enabled. \*/

MAP\_ADCSequenceEnable(ADC0\_BASE, 2);

/\* Clear the interrupt status flag before enabling. This is done to make

\* sure the interrupt flag is cleared before we sample. \*/

MAP\_ADCIntClear(ADC0\_BASE, 2);

MAP\_ADCIntEnable(ADC0\_BASE, 2);

/\* Enable the Interrupt generation from the ADC-0 Sequencer \*/

MAP\_IntEnable(INT\_ADC0SS2);

/\* Enable Timer-0 clock and configure the timer in periodic mode with

\* a frequency of 10 KHz. Enable the ADC trigger generation from the

\* timer-0. \*/

MAP\_SysCtlPeripheralEnable(SYSCTL\_PERIPH\_TIMER0);

while(!(MAP\_SysCtlPeripheralReady(SYSCTL\_PERIPH\_TIMER0)))

{

}

MAP\_TimerConfigure(TIMER0\_BASE, TIMER\_CFG\_A\_PERIODIC);

MAP\_TimerLoadSet(TIMER0\_BASE, TIMER\_A, (systemClock/SAMP\_FREQ));

MAP\_TimerADCEventSet(TIMER0\_BASE, TIMER\_ADC\_TIMEOUT\_A);

MAP\_TimerControlTrigger(TIMER0\_BASE, TIMER\_A, true);

MAP\_TimerEnable(TIMER0\_BASE, TIMER\_A);

//Var Part Start

//Var Part End

while(1)

{

if(setBufProcessReady == true)

{

int16\_t Effect\_FFTIndex[10] = {0};

arm\_cfft\_q15(&arm\_cfft\_sR\_q15\_len4096, (q15\_t \*)dstBufferProcess, IFFTFLAG,

BITREVERSE);

arm\_cmplx\_mag\_q15((q15\_t \*)dstBufferProcess, (q15\_t \*)fftOutput,

NUM\_SAMPLES);

//arm\_max\_q15((q15\_t \*)fftOutput, NUM\_SAMPLES, (q15\_t \*)&setFFTmaxValue,

// &setFFTmaxFreqIndex);

//i32IPart[2] = (int32\_t)((setFFTmaxFreqIndex \* SAMP\_FREQ) / NUM\_SAMPLES);

//i32FPart[2] = (int32\_t)(((setFFTmaxFreqIndex \* SAMP\_FREQ) / NUM\_SAMPLES) \* 1000);

//i32FPart[2] = i32FPart[2] - i32IPart[2] \* 1000;

//for(int index = 1 ; index < NUM\_SAMPLES ; index++)

//{

// UARTprintf("any:%3d,%3d\n", ((int16\_t\*)fftOutput)[index],index);

// //UARTprintf("any:%0d,%3d\n", ((uint32\_t\*)dstBufferNoneProcess)[index],index);

//}

//Find Effect Index

int Effect\_FFTNum = 0;

for(int temp\_index = 1;temp\_index<NUM\_SAMPLES/3;temp\_index++)

{

if(fftOutput[temp\_index]>0)

{

Effect\_FFTIndex[Effect\_FFTNum] = temp\_index ;

Effect\_FFTNum ++ ;

}

}

//Calc THD All

float32\_t THDBuf = (float32\_t)0.0;

float32\_t THD = (float32\_t)0.0;

for(int temp\_index = 1;temp\_index<Effect\_FFTNum;temp\_index++)

{

THDBuf += (float32\_t)fftOutput[Effect\_FFTIndex[temp\_index]] \* (float32\_t)fftOutput[Effect\_FFTIndex[temp\_index]] ;

}

arm\_sqrt\_f32(THDBuf, &THD);

THD = THD/ (float32\_t)fftOutput[Effect\_FFTIndex[0]] ;

//Calc Base Harmonic Freq

float32\_t BasicFreq = 0;

//uint32\_t BasicAmp = 0;

BasicFreq = (float32\_t)Effect\_FFTIndex[0]\* (float32\_t)SAMP\_FREQ / (float32\_t)NUM\_SAMPLES ;

float32\_t Jmp\_Step = (BasicFreq \* (float32\_t)128) / (float32\_t)2 \* (float32\_t)SAMP\_FREQ ;

//BasicAmp = fftOutput[Effect\_FFTIndex[0]] ;

//Calc 1st Harmonic Normalize

float32\_t Normal\_1stAmp = (float32\_t)fftOutput[Effect\_FFTIndex[1]] / (float32\_t)fftOutput[Effect\_FFTIndex[0]] ;

//Calc 3rd Harmonic Normalize

uint32\_t Normal\_2ndAmp = (float32\_t)fftOutput[Effect\_FFTIndex[3]] / (float32\_t)fftOutput[Effect\_FFTIndex[0]];

OLED\_Clear();

delay\_ms(1000) ;

OLED\_Value\_One\_Page(THD,BasicFreq,Normal\_1stAmp,Normal\_2ndAmp) ;

//OLED\_Wave\_showing(dstBufferNoneProcess,1);

delay\_ms(2000) ;

/\*

//int32\_t Test = (int32\_t)(THD \*1000) ;

//int32\_t BasicFreqTest = (int32\_t)(BasicFreq \*1000) ;

for(int index = 1 ; index < NUM\_SAMPLES ; index++)

{

//UARTprintf("any:%3d,%3d\n", BasicFreqTest,Test);

//UARTprintf("any:%3d,%3d\n", ((int16\_t\*)fftOutput)[index],index);

//UARTprintf("any:%3d,%3d,%3d\n", ((int16\_t\*)fftOutput)[index],index,Test);

//UARTprintf("any:%0d,%3d\n", ((uint32\_t\*)dstBufferNoneProcess)[index],index);

}\*/

setBufProcessReady = false ;

}

}

}

附录3：参考文献

[1] SimpleLink™ Ethernet MSP432E401Y MCU LaunchPad™ Development Kit User's Guide (Rev. B)

[2] Dragan Ibrahim ,ESP32 完整开发指南