电赛2023C题

任务

基于 TI 公司的 MCU，设计并制作电感及其品质因数 Q、电容及其损耗角正切 D 的测量装置。被测元件接入，一键启动后，在规定时间内自动完成测量。测量装置要提供专用于监测测试频率的信号输出接口，用于实时监测装置的测试频率，如图 1 所示。

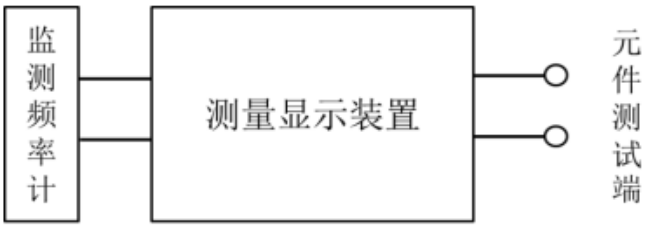

图 1 测量装置结构示意图

基础部分

要求

完成电容量及其损耗角正切 D 的测量。

（1）电容量测量范围：1nF～100nF，测量相对误差的绝对值不大于 5%。

（2）电容 D 值测量范围：0.005～1，测量相对误差的绝对值不大于 5%。

（3）在 1kHz～100kHz 范围内，自定某一固定测量频率。

（4）测量时间不大于 1 秒。

方法

整体思路

原理参考电桥法测电容，通过AD8302幅度相位检测电路，来得到 U_{out} 和 U_{in} 的幅度和相位信息，从而得出所测

遇到的问题

以上方法在理论上一定是可行的，但是在实际中遇到了几个问题

- AD8302的输出和数据手册并不完全一致
  - 具体表现在随增益和相位的变化，曲线并不是完全标准线性的
  - 线性曲线的斜率和手册上不完全一致
  - 各个基准点的输出电压和手册不一致
  - AD8302的输出和输入的频率和幅度都有关系，并非不变的
- 如何采集电压
  - 现在需要测量两路ADC，但是只用过 STM32一路 ADC 单独采集
  - 在中断回调里编写过于复杂的逻辑会导致程序不确定性大
- AD9959总是犯病
  - 有时候波形正常出来，过一会就没了
- 串联电阻阻值不确定
  - 题目要求测量的电容奇形怪状，各个状态都有，不能串联固定的电阻进行测量，如果串联固定电阻，则有时候阻抗角 ϕ 处在极端情况下的时候，就会产生很大的误差

问题的解决

- AD8302
  - 对它进行拟合

    - 具体表现在选定了确定输入信号的频率和幅度，对这个特定的值用信号发生器控制输入，万用表测输出，拟合出真正的**相位-电压**和**增益-电压**曲线
    - 对这个器件的使用还遇到了很大的问题：一是没有搞懂他的输入电压其实最大只能在 $707mv$，导致之前测量的很多远大于这个幅度的拟合曲线实际上都是无效的（当时还奇怪为什么曲线竟然和手册差距这么大，完全不是线性的，现在看来就是因为输入超出了范围，但是竟然没坏，神奇）；二是要搞懂他的增益是哪个通道比哪个通道（$\frac{channel_1}{channel_2}$）；相位是哪个通道减哪个通道，这样子才能利于后边程序的编写
- 采集电压
    - 最开始是用 32 的 ADC 做的，但是一个定时器触发两个 ADC，中断回调函数里写的逻辑过于复杂，导致输出有误
    - 为了使 32 的 ADC 测得更准，我甚至还拟合了 32 的测量值和真实值（万用表测量）直接的关系，得到 `32adc*0.9668=realadc` 的关系
    - 但是最后还是放弃了，一是因为懒得配双重 ADC 了，感觉没什么时间；二是找到了更好的解决方案：直接用 AD8688 来测量输出电压
    - 好处在于：1. AD8688 有多个通道，完全可以同时测量，比配 ADC 简单了不知道多少；2. AD8688 的精度是 16 位，远高于 32 的 12 位，测量非常精确，故最后选用了 AD8688 来进行测量
- AD9959 犯病
    - 没啥解决办法，只能 reset
- 串联电阻阻值不确定
    - 使用光耦继电器控制电路的通断，即可以根据待测电容的特性来确定具体接入电路的是哪一个 R_f 和 R_s
    - 在使用中还遇到了一个问题，我们选用的 R_f 和 R_s 是各有三档，所以 3×3 实际上有 9 种情况，通过程序实现 9 种情况的判断显然不现实，所以固定第一次测量均在中间一档，这样子就剩下了 $2\times2=4$ 种情况

- 但是在使用中发现调整 4 种情况虽然程序好写很多（其实也没有，逻辑判断也挺复杂的），但是会出现一种令人意想不到的情况，举例说明：当增益很大的时候我们肯定要把增益缩小，这个时候我们会调节 R_f 使增益减小，但是如果这个时候我们把 R_s 同步减小了，**最终输出的增益甚至可能比之前的还要大！**
- 于是最后我们就选定了 R_f 和 R_s 同步调节的方法，即同时增大或者减小 R_f 和 R_s，这样就不会出现以上所说的离谱的情况了，而且情况数大大减少，只有三种，而且第一次的测量还是固定在第二档的，所以实际上只有两种
- 但是这里就需要考虑 R_f 和 R_s 的选取问题了，可以通过仿真确定大概合适的范围，再通过实验中具体微调来确定最后最合适的取值
- 实验中还遇到了一个问题，即没有考虑光耦继电器阻值，这个阻值大概是 40Ω，在大挡位没有影响，但是在小挡位就会造成极大的影响（因为小挡位选取的阻值也就是这个数量级的电阻）

最终整体

参数选取

R_f 和 R_s 的三档（RS+光耦继电器阻值）

- RS1：1800+30
- RS2：217.9+37
- RS3：62+39.3
- RF1：4130+30
- RF2：508+18
- RF3：178.9+38.4

拟合斜率

```
#define AD8302_phase_intercept 1.8942

#define AD8302_phase_slope -0.0107

#define AD8302_gain_intercept 0.9065

#define AD8302_gain_slope 0.0304
```

`y=kx+b(y=测量电压；x=相位/dB)`

光耦继电器切换的 C 的判断依据

```
rough_C = cap_D_calculate(real_phase, real_gain);


//第一次测电容，根据测出的电容大小判断是否需要修正

if(rough_C<=5)

{

R1_ON();


recalculate_flag = R1_flag;



calculate_flag = 0;



//重新测量，延时同样是为了等波形稳定
```

```
HAL_Delay(500);

}

else if(rough_C>=30)

{

R3_ON();



recalculate_flag = R3_flag;



calculate_flag = 0;



//重新测量，延时同样是为了等波形稳定

HAL_Delay(500);

}



//判断是否需要重新测量

if (calculate_flag == 0)

{

get_ch1andch2(get_val, real_adc);
```

```
realadc = real_adc[0];

realadc1 = real_adc[1];



real_gain = gain_calculate(realadc);

real_phase = phase_calculate(realadc1);



//重新计算D和C

cap_D_recalculate(real_phase, real_gain);



printf("D最终:%f\r\n", D);

printf("C最终:%f\r\n", C);



}



else if (calculate_flag == 1)

{

//如果不需要切换挡位，则使用之前的数据直接计算

cap_D_calculate(real_phase, real_gain);
```

```
printf("D最终:%f\r\n", D);

printf("C最终:%f\r\n", C);



}
```

其他拟合

```
//计算得出的波形相位关系和实际波形相位关系之间的一次函数关系

#define phase_compensate_intercept 1.257

#define phase_compensate_slope 0.9703
```

这一步是最终额外的拟合，没有原理，纯粹是拟合计算值和真实值直接的误差

最终主函数代码

```
/* USER CODE BEGIN Header */

/**

******************************************************************************
****************

* @file : main.c

* @brief : Main program body
```

```
**********
**********

* @attention

*

* Copyright (c) 2023 STMicroelectronics.

* All rights reserved.

*

* This software is licensed under terms that can be found in
the LICENSE file

* in the root directory of this software component.

* If no LICENSE file comes with this software, it is provided
AS-IS.

*

**********
**********

*/

/* USER CODE END Header */

/* Includes ------------------------------------------
----------*/

#include "main.h"

#include "usart.h"
```

```
#include "gpio.h"



/* Private includes ----------------------------------------
----------------*/

/* USER CODE BEGIN Includes */

#include <stdio.h>

#include <math.h>

#include "tjc_usart_hmi.h"

#include "ad9959.h"

#include "ADS8688.h"

// #include "math.h"

// #include "arm_math.h"

/* USER CODE END Includes */



/* Private typedef -----------------------------------------
----------------*/

/* USER CODE BEGIN PTD */



/* USER CODE END PTD */
```

```
/* Private define ------------------------------------------
---------------*/

/* USER CODE BEGIN PD */



//AD8302 20K幅度1V的相位输出电压和相位差之间的一次函数关系

#define AD8302_phase_intercept 1.8942

#define AD8302_phase_slope -0.0107

#define AD8302_gain_intercept 0.9065

#define AD8302_gain_slope 0.0304



//计算得出的波形相位关系和实际波形相位关系之间的一次函数关系

#define phase_compensate_intercept 1.257

#define phase_compensate_slope 0.9703



//光耦切换时所参考数据的上下限，原理见电桥法测电容

//D和Rs相关,如果测量的D很小则需要串联Rs，反之不需要串联

//Av和Rf相关，如果Av很小，则需要提高Rf放大输出电压，反之不需要

//以上操作均为避免极端情况，以提高测量精度

#define RS_LOW_LINE 0.26794 //tan(15)
```

```
#define RS_HIGH_LINE 3.7321 //tan(75)

#define RF_LOW_LINE 0.1

#define RF_HIGH_LINE 3



//RS和RF阻值(RS/RF+光耦继电器阻值)

#define RS1 1800+30//4020+30

#define RS2 217.9+37//217470+37

#define RS3 62+39.3//61.7+39.3

#define RF1 4130+30//9120+30

#define RF2 508+18//1020+18

#define RF3 178.9+38.4//178.6+38.4



//RS和RF判断标志位，挡位2设置成0是为了方便判断是否进行了挡位的更改

#define R1_flag 1

#define R2_flag 2

#define R3_flag 3
```

```
/* USER CODE END PD */



/* Private macro -------------------------------------------
---------------*/

/* USER CODE BEGIN PM */



/* USER CODE END PM */



/* Private variables ---------------------------------------
---------------*/



/* USER CODE BEGIN PV */



//相位和增益的adc采集次数

int phaseadctime = 0;

int gainadctime = 0;

float sum = 0;

float sum1 = 0;
```

```
//32adc采集的真实值
float realadc = 0;
float realadc1 = 0;

float real_adc[2];

//AD8688测量数据
uint16_t get_val[2];



//计算的相位分别存在三个数组里，三次采集相位算平均
float phase;
float dB;

//挡位切换重新计算标志位，默认挡位2是2
int recalculate_flag = 2;
```

```
//一次计算完成标志位

int calculate_flag = 1;



//计算D和C值

float D;

float C;



//第一次粗测C的值

float rough_C;



//实际相位增益大小

float real_phase;

float real_gain;

/* USER CODE END PV */



/* Private function prototypes -----------------------------
---------------*/

void SystemClock_Config(void);

/* USER CODE BEGIN PFP */
```

```
//AD8302 20K同相的输出电压和增益比之间的二次函数关系为y = 0.0287x2 - 0.2568x + 1.1612，其中y为输出电压，x为增益倍数

//计算相位大小

float phase_calculate(float realadc);



//计算增益大小

float gain_calculate(float realadc);



//第一次计算电容D和C大小：中间挡位

float cap_D_calculate(float phase, float gain);



//重新计算电容D和C大小：两边挡位

float cap_D_recalculate(float phase, float gain);



//光耦继电器控制

void R1_ON();

void R2_ON();

void R3_ON();
```

```
/* USER CODE END PFP */



/* Private user code ---------------------------------------
----------------*/

/* USER CODE BEGIN 0 */



/* USER CODE END 0 */



/**

* @brief The application entry point.

* @retval int

*/

int main(void)

{



/* USER CODE BEGIN 1 */
```

```
/* USER CODE END 1 */



/* MCU Configuration--------------------------------------------
--------------*/



/* Reset of all peripherals, Initializes the Flash interface
and the Systick. */

HAL_Init();



/* USER CODE BEGIN Init */



/* USER CODE END Init */



/* Configure the system clock */

SystemClock_Config();



/* USER CODE BEGIN SysInit */



/* USER CODE END SysInit */
```

```
/* Initialize all configured peripherals */

MX_GPIO_Init();

MX_USART1_UART_Init();

MX_USART2_UART_Init();

/* USER CODE BEGIN 2 */

initRingBuff();

HAL_UART_Receive_IT(&huart2, RxBuff, 1);



//ad9959初始化，控制输出幅度200MV，频率20k

ad9959_init();

ad9959_write_amplitude(AD9959_CHANNEL_1, 736);

ad9959_write_frequency(AD9959_CHANNEL_1, 20000);

HAL_Delay(500);



//ADS8688初始化

ADS8688_Init_Mult();



/* USER CODE END 2 */
```

```
/* Infinite loop */

/* USER CODE BEGIN WHILE */

//首先调到第二档

R2_ON();



//延时一会，等到波形稳定再进行测量和计算，这个时间为了保险这里取大一点

HAL_Delay(500);



//AD8688测电压

get_ch1andch2(get_val, real_adc);



realadc = real_adc[0];

realadc1 = real_adc[1];



real_gain = gain_calculate(realadc);

real_phase = phase_calculate(realadc1);
```

```
//粗测C，作为挡位调整判断依据

rough_C = cap_D_calculate(real_phase, real_gain);



//第一次测电容，根据测出的电容大小判断是否需要修正

if(rough_C<=5)

{

R1_ON();



recalculate_flag = R1_flag;



calculate_flag = 0;



//重新测量，延时同样是为了等波形稳定

HAL_Delay(500);

}

else if(rough_C>=30)

{

R3_ON();
```

```
recalculate_flag = R3_flag;



calculate_flag = 0;



//重新测量，延时同样是为了等波形稳定

HAL_Delay(500);

}









//判断是否需要重新测量

if (calculate_flag == 0)

{

get_ch1andch2(get_val, real_adc);



realadc = real_adc[0];

realadc1 = real_adc[1];
```

```
real_gain = gain_calculate(realadc);

real_phase = phase_calculate(realadc1);



//重新计算D和C

cap_D_recalculate(real_phase, real_gain);



printf("D最终:%f\r\n", D);

printf("C最终:%f\r\n", C);



}



else if (calculate_flag == 1)

{

//如果不需要切换挡位，则使用之前的数据直接计算

cap_D_calculate(real_phase, real_gain);



printf("D最终:%f\r\n", D);

printf("C最终:%f\r\n", C);
```

```
}

while (1) {

}

/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

/* USER CODE END 3 */
}

/**
* @brief System Clock Configuration
* @retval None
*/
void SystemClock_Config(void)
```

```
{

RCC_OscInitTypeDef RCC_OscInitStruct = {0};

RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};



/** Configure the main internal regulator output voltage

*/

__HAL_RCC_PWR_CLK_ENABLE();

__HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);



/** Initializes the RCC Oscillators according to the specified
parameters

* in the RCC_OscInitTypeDef structure.

*/

RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;

RCC_OscInitStruct.HSEState = RCC_HSE_ON;

RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;

RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;

RCC_OscInitStruct.PLL.PLLM = 4;

RCC_OscInitStruct.PLL.PLLN = 168;
```

```
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;

RCC_OscInitStruct.PLL.PLLQ = 4;

if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)

{

Error_Handler();

}



/** Initializes the CPU, AHB and APB buses clocks

*/

RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;

RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;

RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;

RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;

RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;



if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5)
!= HAL_OK)

{
```

```
Error_Handler();

}

}



/* USER CODE BEGIN 4 */



float phase_calculate(float realadc)

{



//用公式计算相位差

phase = (realadc - AD8302_phase_intercept) /
AD8302_phase_slope;



printf("real_phase:%f\r\n", phase);



return phase;



}
```

```
float gain_calculate(float realadc)

{

dB = (realadc - AD8302_gain_intercept) / AD8302_gain_slope;



//通过dB算增益

float realgain = pow(10, dB / 20);

printf("averagegain:%f\r\n", realgain);



return realgain;







}



void C_Display(void)

{

TJCPrintf("main.t4.txt=\"测量结束\"");

}
```

```
void L_Display(void)

{

TJCPrintf("L.t4.txt=\"测量结束\"");

}



void TJC_USART(void)

{

while (usize >= 6) {

// 校验帧头帧尾是否匹配

if (u(0) != 0x55 || u(3) != 0xff || u(4) != 0xff || u(5) !=
0xff) {

// 不匹配删除1字节

udelete(1);

} else {

// 匹配，跳出循环

break;

}

}
```

```
// 进行解析

if (usize >= 6 && u(0) == 0x55 && u(3) == 0xff && u(4) == 0xff

&& u(5) == 0xff) {

if ((u(1) == 0x00)) { // 电容测量

// C_auto_measure();

C_Display();

}

if (u(1) == 0x01) { // 电感测量

// L_auto_measure();

L_Display();

}

udelete(6);

}

}



float cap_D_recalculate(float phase, float gain)

{

int RS, RF = 0;

//初始化RS和RF的值
```

```
switch (recalculate_flag)

{

case R1_flag:

RF = RF1;

RS = RS1;

break;

case R2_flag:

RF = RF2;

RS = RS2;

break;

case R3_flag:

RF = RF3;

RS = RS3;

break;

default:

break;

}
```

```
//进行计算

float Zx = RF / gain;

float Xc = Zx * sin((M_PI * phase / 180));

float Rx = Zx * cos((M_PI * phase / 180)) - RS;



C = 1000000000 / (2 * M_PI * 20000 * Xc);




if(C>30)

{

//经验修正C

C = C - 0.07*C;

Xc = 1000000000/(2*M_PI*20000*C);

//经验修正Rx

Rx = Rx -5;

D = Rx/ Xc;

}
```

```
//极端情况D测不准，手动修正

else if((1.5<C)&(C<2))

{

D = 0.012131;

}

else if(C<1.4)

{

D = 0.035078;

}



return C;

}



float cap_D_calculate(float phase, float gain)

{

//初始化RS和RF的值

int RS = RS2;

int RF = RF2;
```

```
//进行计算

float Zx = RF / gain;

float Xc = Zx * sin((M_PI * phase / 180));

float Rx = Zx * cos((M_PI * phase / 180)) - RS;

D = Rx / Xc;

C = 1000000000 / (2 * M_PI * 20000 * Xc);



return C;

}



void R1_ON()

{

HAL_GPIO_WritePin(LIGHT_RS1_GPIO_Port, LIGHT_RS1_Pin,
GPIO_PIN_SET);

HAL_GPIO_WritePin(LIGHT_RS2_GPIO_Port, LIGHT_RS2_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RS3_GPIO_Port, LIGHT_RS3_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF1_GPIO_Port, LIGHT_RF1_Pin,
GPIO_PIN_SET);
```

```
HAL_GPIO_WritePin(LIGHT_RF2_GPIO_Port, LIGHT_RF2_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF3_GPIO_Port, LIGHT_RF3_Pin,
GPIO_PIN_RESET);

}

void R2_ON()

{

HAL_GPIO_WritePin(LIGHT_RS1_GPIO_Port, LIGHT_RS1_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RS2_GPIO_Port, LIGHT_RS2_Pin,
GPIO_PIN_SET);

HAL_GPIO_WritePin(LIGHT_RS3_GPIO_Port, LIGHT_RS3_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF1_GPIO_Port, LIGHT_RF1_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF2_GPIO_Port, LIGHT_RF2_Pin,
GPIO_PIN_SET);

HAL_GPIO_WritePin(LIGHT_RF3_GPIO_Port, LIGHT_RF3_Pin,
GPIO_PIN_RESET);

}

void R3_ON()

{
```

```
HAL_GPIO_WritePin(LIGHT_RS1_GPIO_Port, LIGHT_RS1_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RS2_GPIO_Port, LIGHT_RS2_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RS3_GPIO_Port, LIGHT_RS3_Pin,
GPIO_PIN_SET);

HAL_GPIO_WritePin(LIGHT_RF1_GPIO_Port, LIGHT_RF1_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF2_GPIO_Port, LIGHT_RF2_Pin,
GPIO_PIN_RESET);

HAL_GPIO_WritePin(LIGHT_RF3_GPIO_Port, LIGHT_RF3_Pin,
GPIO_PIN_SET);

}



/* USER CODE END 4 */



/**

* @brief This function is executed in case of error
occurrence.

* @retval None

*/

void Error_Handler(void)
```

```
{

/* USER CODE BEGIN Error_Handler_Debug */

/* User can add his own implementation to report the HAL error
return state */

__disable_irq();

while (1) {

}

/* USER CODE END Error_Handler_Debug */

}



#ifdef USE_FULL_ASSERT

/**

* @brief Reports the name of the source file and the source
line number

* where the assert_param error has occurred.

* @param file: pointer to the source file name

* @param line: assert_param error line source number

* @retval None

*/

void assert_failed(uint8_t *file, uint32_t line)
```

```
{

/* USER CODE BEGIN 6 */

/* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

/* USER CODE END 6 */

}

#endif /* USE_FULL_ASSERT */
```