

---

# LPC54606平台的周期信号基波幅值提取程序使用文档

---

LPC54606平台的周期信号基波幅值提取程序使用文档

## 一、引言

- 1、说在前面
- 2、改用此程序的缘由
- 3、硬件提取幅值和软件FFT提取幅值在程序上的优缺点
  - (1)硬件优点
  - (2)硬件缺点
  - (3)软件优点
  - (4)软件缺点

## 二、原理

- 1、架构
- 2、FFT数学原理
- 3、程序原理
  - (1)向用户开放的接口函数
    - Fourier\_Init
    - Fourier\_Once
  - (2)不向用户开放的内部函数（如果仅仅是使用无需关心内部如何实现）
    - Fourier\_dma\_reload
    - Fourier\_adc\_reload
    - IRQ\_Ctrl\_ADC\_DMA
    - Fourier\_DMAReadBuff
    - FFT
    - kalmanFilter

## 三、使用指南

- 1、使用步骤（非常重要！！！！）
- 2、注意事项（非常重要！！！！）

## 四、问题汇总和部分解决办法

- 1、读取数据时卡住
  - 2、数据有延迟或抖动厉害
-

# 一、引言

---

## 1、说在前面

虽然目前做电磁的不多，但是后期真正做比赛时，电磁还是必不可少的，因此能有一个稳定完善的电磁方案还是非常重要的，由于这个程序并不完善，是否稳定还需实际应用测试。

在编写这个底层的时候，我踩到的坑太多太多，需要解决的问题和之前做智能车遇到的完全不同，就像是白手起家，我不想学弟学妹在未来移植和完善这个程序的时候，依然要解决相同的问题，因此我写下这篇说明，记录我程序的流程和遇到的困难，希望能帮助你们理解和运用本程序。

因为理论和实际往往会相差很大，这里我需要你们务必趁早使用这个程序，调试过程中我遇到了无数次的数组越界，堆栈溢出的情况，因为能力限制我至今未确认真正原因，不过很大可能性是DMA传输和FFT迭代过程出了问题。这个问题在今后调车会是致命的，他会直接导致程序卡死，并且无法恢复，只能重新复位。

所以这就需要你们在训练阶段进行大量的测试，需要每个人的优化，如果遇到任何有关这段程序的问题，一定要趁早解决，并且告诉我。

## 2、改用此程序的缘由

需要采集到的电磁信号是一个20Khz的正弦信号，且信号的峰峰值与电感和电磁线的距离呈倍数关系

传统的办法是利用预先设计好的滤波电路将20Khz的正弦信号转换为直流信号，再通过单片机AD转换直接得到信号的强度，但是如果精度要求比较高，会增加电路的复杂程度，并且由于电磁前瞻板的空间和体积限制，电路板不能做的很大，如果没有扎实的焊接基础和电路基础，很难一次性焊接成功，且后续的电路检修也非常困难。

因此就萌生了这样一个想法，直接通过简单的电路将信号一次放大后输入单片机，通过程序来分析正弦信号的峰峰值，这也就大大减轻了硬件上的压力，也降低了电路板出现问题的可能性。

## 3、硬件提取幅值和软件FFT提取幅值在程序上的优缺点

### (1)硬件优点

程序实现简单，只需使用AD转换测出电压即可，RAM消耗极低，不占用芯片资源，没有芯片平台的限制

### (2)硬件缺点

滤波电路复杂，不容易焊接，检修困难，对电路的设计者要求较高，元器件精度要求较高

(3)软件优点

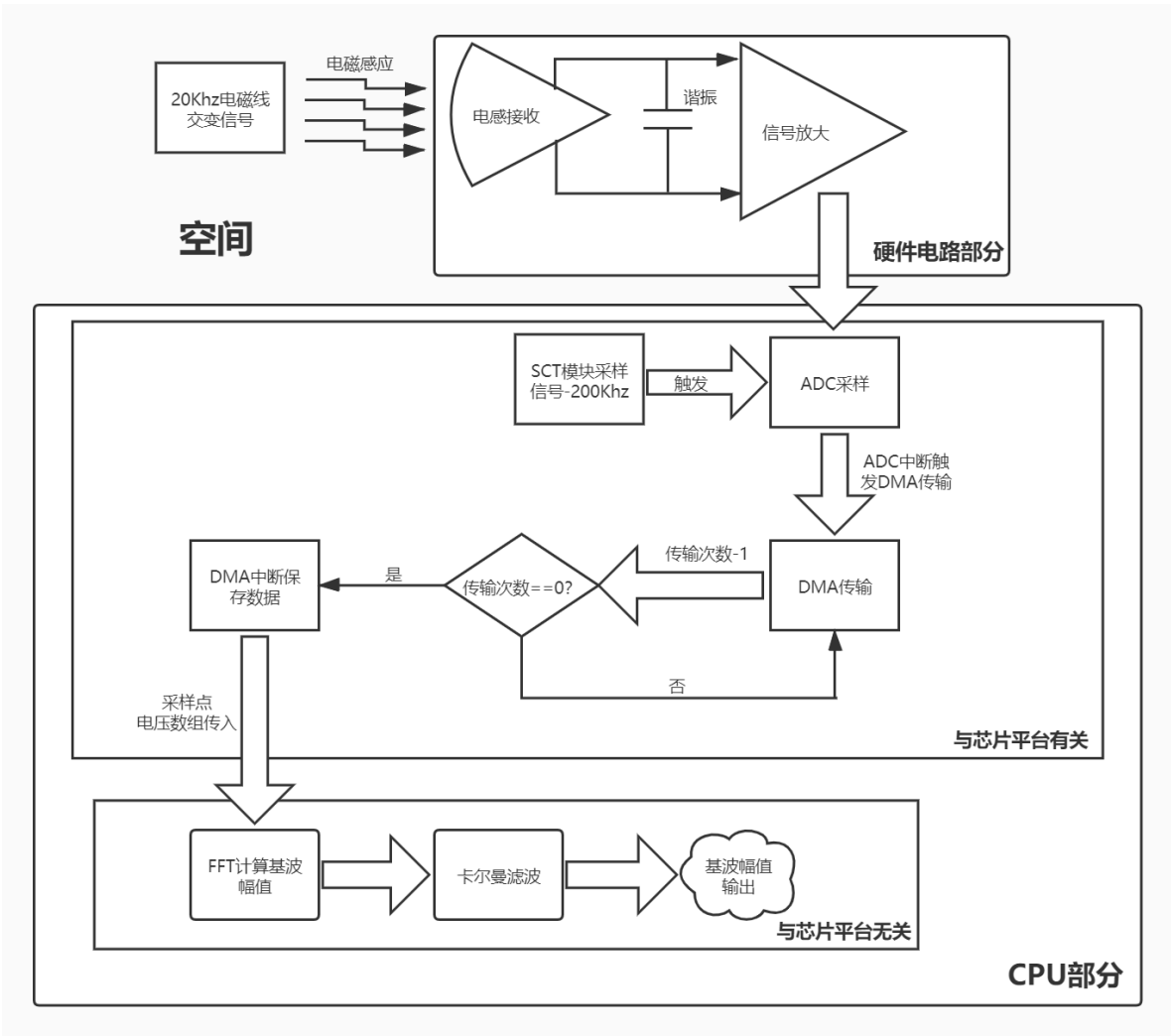
所需电路非常简单，且对电路的稳定性和元器件的精度要求没有硬件这么高，由于使用的是傅里叶算法，他只会提取到信号内的20Khz基波幅值，自动滤除其他的干扰信号，在底层编写正确的情况下，软件提取一次基波幅值只需115us左右，并不比硬件的时间长很多。最终的结果添加了卡尔曼滤波，保证了在信号不变，12bit的ADC最高精度下，最终结果波动不超过0.2%，且信号跟随性比较好，不同通道切换没有很大的干扰。

(4)软件缺点

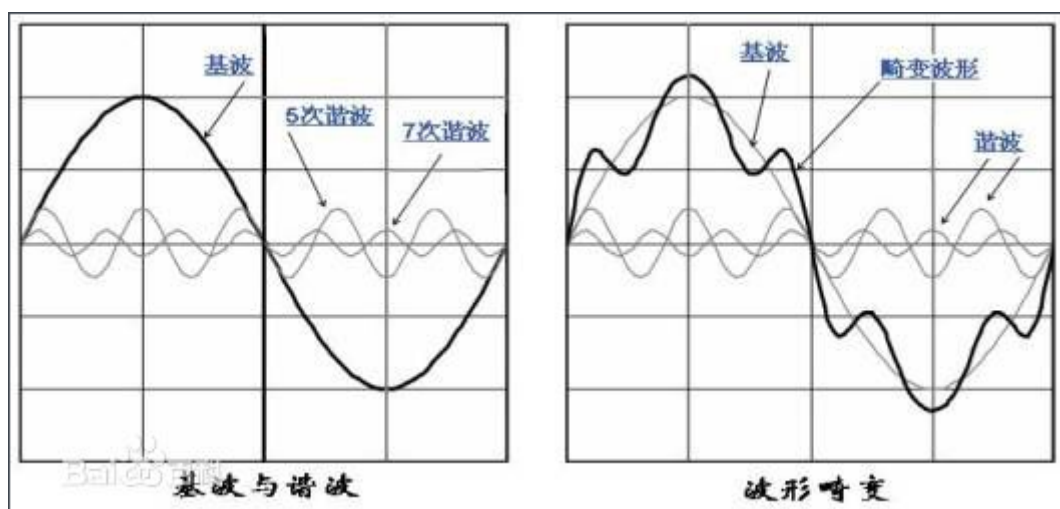
所需芯片RAM较高，且移植比较困难，由于使用的是DMA直接传输，较容易造成不明原因的数组越界，程序是否稳定需要比较多的测试，这也是为什么我写下这篇教程，是为了能让使用到它的同学能比较快的查找到错误原因，也能辅助使用者完善本程序。

二、原理

1、架构



## 2、FFT数学原理



假设被检测信号如以下公式所示

$$u = \sum_{n=1}^{\infty} A_n \cos(n\omega t + \varphi_n)$$

其中， $A_n$ 、 $n\omega$ 、 $\varphi_n$  为  $n$  次谐波的幅值、角频率、初始相位， $\omega = 2\pi f$ ， $f$  为基波频率。

在离散系统中，若采样频率为  $f_s$ ，则采样周期  $T_s = 1/f_s$ ，每一个基波周期的采样点数为  $N = f_s / f$ ，按照离散傅里叶变换的基本原理，在  $t$  时刻  $u(t)$  可以表示为以下公式，其中  $u(t)$  为某个采样时刻的电压值

$$u(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega t) + \sum_{n=1}^{\infty} b_n \sin(n\omega t)$$

其中：

$$a_0 = \frac{1}{N} \sum_{t=t_0}^{t_0+(N-1)T_s} u(t)$$

$$a_n = \frac{2}{N} \sum_{t=t_0}^{t_0+(N-1)T_s} u(t) \cos(n\omega t)$$

$$b_n = \frac{2}{N} \sum_{t=t_0}^{t_0+(N-1)T_s} u(t) \sin(n\omega t)$$

根据上述公式所示，以  $n$  次谐波电压为例，可得其电压的傅里叶级数表达式如以下公式所示

$$u_n(t) = a_n \cos(n\omega t) + b_n \sin(n\omega t) =$$

$$\sqrt{a_n^2 + b_n^2} \left[ \frac{a_n}{\sqrt{a_n^2 + b_n^2}} \cos(n\omega t) + \frac{b_n}{\sqrt{a_n^2 + b_n^2}} \sin(n\omega t) \right] =$$

$$\sqrt{a_n^2 + b_n^2} [\cos \varphi_n \cos(n\omega t) - \sin \varphi_n \sin(n\omega t)] =$$

$$\sqrt{a_n^2 + b_n^2} \cos(n\omega t + \varphi_n) \quad (10)$$

其中  $A_n = \sqrt{a_n^2 + b_n^2}$  即为  $n$  次谐波电压的幅值

因此,  $\cos(n\omega t)$  和  $\sin(n\omega t)$  便可以转化为一个信号周期中的采样点。以基波信号为例, 正弦  $\sin(\omega t)$  和余弦  $\cos(\omega t)$  分为  $N$  个采样点, 由于  $\sin(\omega t)$  和  $\cos(\omega t)$  是以  $2\pi/\omega$  为周期的函数, 因此在  $k = N + 1$  时, 可令  $k = 1$  重新开始计数, 从而使得  $\sin(\omega t)$  和  $\cos(\omega t)$  一直反复利用  $N$  点采样值循环计算。

**通俗来说**, 如果需要计算一个 20KHz 正弦信号的幅值, 只需采集到他的一个完整周期, 例如 10 个采样点, 那么这个信号的幅值即为

每个采样点的电压  $u(t)$  乘以  $\sin(2\pi f \cdot (t \cdot T_s))$ , 再乘以  $2/N$  ----- 得到  $a_n$

每个采样点的电压  $u(t)$  乘以  $\cos(2\pi f \cdot (t \cdot T_s))$ , 再乘以  $2/N$  ----- 得到  $b_n$

( $f$  为基波频率, 这里为 20KHz,  $T_s$  为采样周期, 这里为  $1/200\text{KHz}$ , 即为  $5\mu\text{s}$ ,  $N$  为采样点数目, 这里为 10)

最后根据公式  $A_n = \sqrt{a_n^2 + b_n^2}$  得到本信号的基波幅值  $A_n$

由于公式中的  $2\pi f \cdot (t \cdot T_s)$  除了  $T$  是变量外, 其他都为常量, 且  $t$  是以 0-10 规律变化, 因此可以提前计算好  $\sin$  和  $\cos$  的值, 最后只需用每个采样点的电压乘以预先计算好的  $\sin$  和  $\cos$  即可, 以加快运算速度。

### 3、程序原理

#### (1) 向用户开放的接口函数

----- `Fourier_Init`

```
void Fourier_Init(DMACH_enum dmach, ADCCH_enum ch);  
/**  
 * 基波幅值提取初始化  
 * @param DMACH_enum dmach 设置不同ADC端口数据传输的DMA通道号, 范围DMA_CH0~DMA_CH29  
 *        ADCCH_enum ch      设置AD转换端口  
 * @return 无  
 * @example Fourier_Init(DMA_CH0, ADC_CH6_B0); 设置B0口的基波幅值提取方式, 数据传输用DMA_CH0通道  
 */
```

----- `Fourier_Once`

```
uint16 Fourier_Once(ADCCH_enum ch, ADCRES_enum resolution);  
/**  
 * 计算一个通道的一次基波幅值  
 * @param ADCCH_enum ch      设置AD转换端口  
 *        ADCRES_enum resolution 设置AD转换位数, 范围6位, 8位, 10位, 12位  
 * @return 卡尔曼滤波以后的基波幅值  
 * @example FFT(Data_Buff) 计算Data_Buff数组内的基波幅值  
 */
```

#### (2) 不向用户开放的内部函数 (如果仅仅是使用无需关心内部如何实现)

----- `Fourier_dma_reload`

```

__STATIC_INLINE void Fourier_dma_reload(DMACH_enum dmach, void *SADDR, void
*DADDR, uint16 count);
/**
 *DMA参数重载（内部函数，无需调用）
 *使用__STATIC_INLINE为了将这段函数内嵌到使用该函数的地方，这样可以减少函数调用的时间
 *@param DMACH_enum dmach 设置传输的DMA通道号,范围DMA_CH0~DMA_CH29
 *      void *SADDR          DMA传输源地址
 *      void *DADDR          DMA传输目的地址
 *      uint16 count          传输次数
 *@return 无
 *@example Fourier_dma_reload(DMA_CH0, (void*)&ADC0->SEQ_GDAT[0],
(void*)&Buff[0],20); 将ADC结果寄存器内的数据传输至Buff数组内，循环20次
 */

```

#### ----- Fourier\_adc\_reload

```

__STATIC_INLINE void Fourier_adc_reload(ADCCH_enum ch,ADCRES_enum resolution);
/**
 *ADC参数重载（内部函数，无需调用）
 *使用__STATIC_INLINE为了将这段函数内嵌到使用该函数的地方，这样可以减少函数调用的时间
 *@param ADCCH_enum ch      设置AD转换端口
 *      ADCRES_enum resolution 设置AD转换位数，范围6位，8位，10位，12位
 *@return 无
 *@example Fourier_adc_reload(ADC_CH6_B0,ADC_12BIT);设置B0口的AD转换参数，并设置为12
位ADC
 */

```

#### ----- IRQ\_Ctrl\_ADC\_DMA

```

__STATIC_INLINE void IRQ_Ctrl_ADC_DMA(uint8 ONorOFF);
/**
 *中断控制（内部函数，无需调用）
 *使用__STATIC_INLINE为了将这段函数内嵌到使用该函数的地方，这样可以减少函数调用的时间
 *@param uint8 ONorOFF  中断开关，1为开，0为关，控制DMA和ADC中断
 *@return 无
 *@example IRQ_Ctrl_ADC_DMA(1); 打开ADC和DMA中断
 */

```

#### ----- Fourier\_DMAReadBuff

```

void Fourier_DMAReadBuff(ADCCH_enum ch,ADCRES_enum resolution,uint16 *Buff_Out);
/**
 *获取N此电压数据至指定数组内
 *@param ADCCH_enum ch      设置AD转换端口
 *      ADCRES_enum resolution 设置AD转换位数，范围6位，8位，10位，12位
 *      uint16 *Buff_Out  传入N个采样点电压数据数组的指针
 *@return 无
 *@example Fourier_DMAReadBuff(ADC_CH6_B0,ADC_12BIT,Data_Buff); 将B0口的N个采样点的
12位电压数据传入Data_Buff数组内
 */

```

#### ----- FFT

```
double FFT(uint16 *adc_data);
/**
 *FFT基波幅值提取函数
 *@param uint16 *adc_data    传入保存了N个采样点的电压数据
 *@return 基波幅值，如果传入的电压数据为6位的，返回基波幅值范围为0-31
 *          如果传入的电压数据为8位的，返回基波幅值范围为0-127
 *          如果传入的电压数据为10位的，返回基波幅值范围为0-512
 *          如果传入的电压数据为12位的，返回基波幅值范围为0-2048
 *@example FFT(Data_Buff) 计算Data_Buff数组内的基波幅值
 */
```

## ----- kalmanFilter

```
float kalmanFilter(struct kalman_Data_t* KALMAN,float input);
/**
 *卡尔曼滤波器
 *@param struct kalman_Data_t* KALMAN 卡尔曼结构体参数
 *          float input 需要滤波的参数的测量值（即传感器的采集值）
 *@return 滤波后的参数（最优值）
 */
```

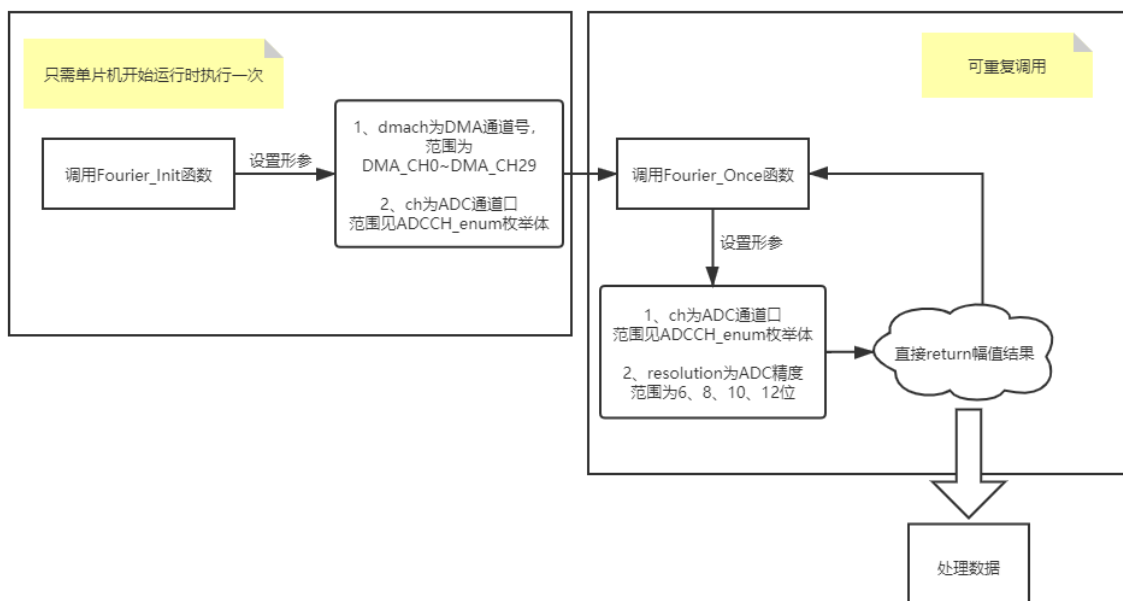
## 三、使用指南

### 1、使用步骤（非常重要！！！！）

- (1) 将Fourier\_DMA.c和Fourier\_DMA.h文件添加进工程文件夹内，并且设置好头文件路径
- (2) 将Fourier\_interrupt\_Func()函数添加进isr.c文件内的中断服务函数DMA0\_DriverIRQHandler

```
void DMA0_DriverIRQHandler(void)
{
    Fourier_interrupt_Func();
}
```

### (2) 看图使用







### 解决办法：

1) 在FFT进行累加运算前尽量的缩小每次迭代的值的大小，具体办法是例如可以在做三角函数表的时候，对每个数据除以10，在最终开完平方并且开根号了以后，再乘以10，这样也许会减小迭代过程中的数据溢出可能性，但也会损失精度。

2) 在H文件内，降低ADC\_Samp\_num，即每个周期的采样点数，可以直接降低迭代次数防止溢出，但是修改这里的话需要重新计算cos\_sin\_search三角函数查表，比较麻烦，且会降低精度，不建议使用，但他是一定能够解决数据溢出问题。

### (2) DMA传输目的寄存器在其他重要数据寄存器的附近（主要原因）

**原因：**之前我是使用了结构体的方式来保存数据缓冲区，标志位，结果等等数据，由于结构体内各变量在单片机内部所占的是一片连续的存储空间。我的猜想是，DMA搬运数据也不是百分百正常，会出现偶尔的将数据传输至目标数组附近的一些数据寄存器内，这个时候如果旁边恰好是某个很重要的标志位，那么就会导致这个标志位被错误的置位或清零，会造成整个程序逻辑的错乱。例如程序中的START\_Flag是一个非常重要的标志位，如果他没在该被清零的时候清零，会导致程序一直卡住，也是导致读取数据卡住的一个主要原因。

### 解决办法：

1) 将Fourier\_Once函数内部最后返回的值改为Result\_WithoutFilt，并且屏蔽滤波程序，看是否会再出现问题

2) 慎用结构体，需要在详细了解了结构体在内存中的占用方式后再去合理使用他，结构体固然好，但是也不能滥用，能用指针传递参数的时候就最好不要使用全局变量和结构体。

3) 在结构体内各个数据间添加空的变量空间来隔离数据。

```
struct Fourier_Data_t
{
    uint8 reserve0[100];           //保留位，防止附近寄存器溢出导致的数据异常

    DMACH_enum DMA_CH[14];        //保存每个ADC通道的DMA传输通道

    uint8 reserve1[100];           //保留位，防止附近寄存器溢出导致的数据异常

    ADCCH_enum ADCCH_Save;        //保存此时正在转换的通道

    uint8 reserve2[100];           //保留位，防止附近寄存器溢出导致的数据异常

    uint8 Busy_Flag;              //全局忙标志

    uint8 reserve3[100];           //保留位，防止附近寄存器溢出导致的数据异常
}Fourier_Data;
```

例如上面的的结构体定义，在各个数据之间添加了reserve保留区的方式来隔离重要变量。

当然这是万不得已的笨办法，它会浪费RAM空间，不到迫不得已最好不要使用这种方法。

4) 有时会有出现局部变量的溢出也会导致卡住，这个问题的原因我到现在还没有搞明白，暂且把他归为玄学哈哈哈哈哈，这也是为何我需要你们帮助我做大量的测试。

## 2、数据有延迟或抖动厉害

**原因：**滤波算法或卡尔曼滤波的参数问题

本来这个最后的滤波我是选用滑动滤波的，但是鉴于滑动滤波所占RAM比较大，而这个程序又对数据溢出异常敏感，因此最终我选择了卡尔曼滤波，这是一个非常优秀的滤波算法。

#### **解决办法：**

改动Fourier\_Init函数内初始化的参数kalman\_Data[ch].R,

这个参数调大了会造成数据跟随性变差，简单来说数据会有延迟，但是抖动会变弱，但是参数调小了数据跟随性会变好，但是抖动会变得厉害。

我调出的值是0.2，也并不是仔细调整出来的，你们可以根据实际需要改动参数，