中文摘要

本系统设计以 TI 的功率放大器芯片 TPA3112D1 为核心构建一个音频功率放大器，加入了啸叫检测与抑制功能，可实现对台式麦克风音频信号的放大并通过喇叭输出。设计的系统主要由拾音电路模块、功率放大电路模块、啸叫检测和啸叫抑制模块四个模块组成，完成了音频功率放大器的简单智能实现。拾音电路模块采用分立元件三极管实现信号的采集及放大；啸叫检测设计上采用 C 语言编程，实现了啸叫频率和相应的功率放大器输出功率的计算算法，对测试过程中产生的啸叫进行实时监测并显示相应频率及输出功率；啸叫的抑制由均衡器实现，可有效抑制频率范围在 200Hz ~ 10kHz 的啸叫。

关键词：TPA3112D1；麦克风；啸叫

目录

1．系统方案设计与论证

1.1 系统设计与实现方法

1.1.1 设计要求

基于 TI 的功率放大器芯片 TPA3112D1，设计并制作一个带啸叫检测与抑制功能的音频放大器，完成对台式麦克风音频信号进行放大，通过功率放大电路送喇叭输出。电路示意图如图 1 所示。

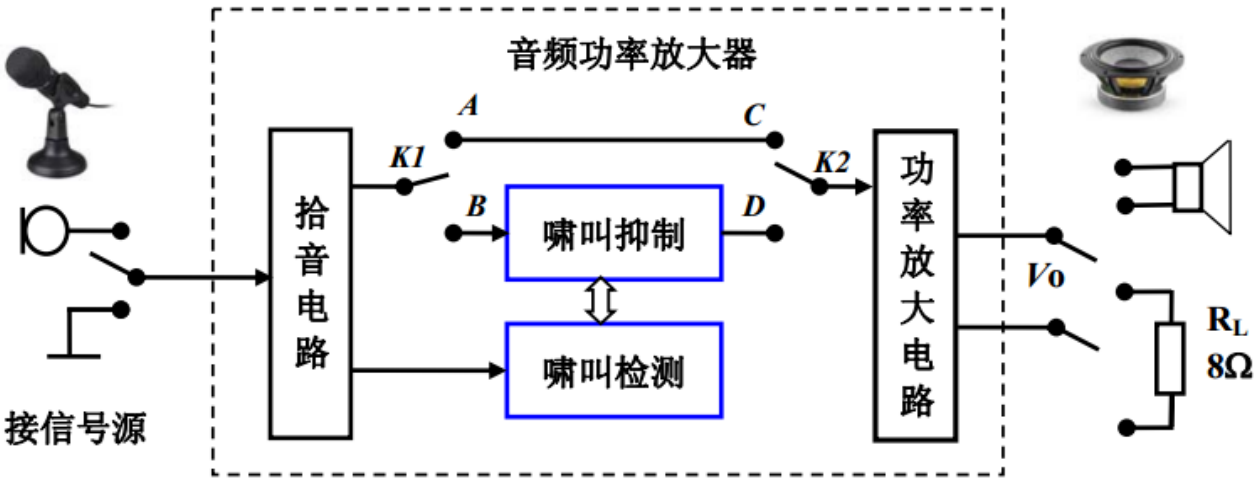


图 1　电路示意图

1.1.2 设计思路

由图 1 可见，该电路宜采用模块化的设计思想。系统可大致分为四个主要模块：拾音电路模块、功率放大电路模块、啸叫检测模块和啸叫抑制模块。

在功率放大之前，需要有一定强度的信号来推动功率放大器，因此拾音电路的本质即为前置放大级电路，将弱信号进行电压放大；功率放大电路则实现对信号的电压和电流放大任务，需要基于功率放大器芯片 TPA3112D1 设计；产生啸叫的必要条件有过载量、距离、角度、频率，因而利用这四个条件是检测和抑制啸叫的关键。由于啸叫检测电路要求能实时监测所产生啸叫并计算啸叫的频率，我们只能利用软件来实现“啸叫”频率的实时检测；抑制或消除啸叫要从啸叫的四个原因入手，只要破坏其中任一个条件，就可达到目的。

1.2 模块方案选择与论证

1.2.1 拾音电路模块方案选择与论证

(1) 方案一：利用运算放大器

拾音电路可以采用集成运算放大器构成的前置放大器，采用同相放大电路结构，电阻的放大倍数由反馈电阻和反相端电阻决定，但跟随电路需要具有输入电阻大，输出电阻小的特点，成本较高。

(2) 方案二：利用三极管

拾音电路也可以采用低噪的三极管实现小信号的放大，由集电极电流和等效负载电阻的大小决定其放大倍数，成本低。

系统输入的小信号需要通过麦克风采集得到，不同于直接给定 mV 级输入信号的前置放大电路，通过麦克风较难采集到信号源产生的信号，而且拾取到的音频信号较小，需要有足够大的放大倍数的放大电路对其进行放大，并且方案二的成本低。综合考虑，拾音电路模块选择方案二，以实现信号源信号的拾取及麦克风音频信号的放大。

1.2.2 功率放大电路模块方案论证

系统基于 TI 的功率放大器芯片 TPA3112D1 来进行设计。TPA3112D1 是一款具有 SpeakerGuard^{TM} 的 25W 单声道、无需外加滤波器的 D 类音频放大器。该芯片供电范围为 8V~26V；采用 H 桥作为功率输出级，使得其可在输出没有传统的 LC 滤波器的情况下直接驱动感性负载；输入的音频信号可以是差分形式，其中在 24V 供电情况下，满负载驱动 8Ω 的桥接式扬声器，声音失真率仅为 0.1%。[1]

根据 TPA3112D1 的工作原理及芯片管脚设计 TPA3112D1 外围电路，实现音频功率放大。

TPA3112D1 外围电路的设计包括：供电电路设计、音频输入电路设计、功率输出电路设计、功率限制引脚外部电路设计等，具体电路设计请参见文献[1]，在此不再赘述。

1.2.3 啸叫检测模块方案论证

啸叫检测设计上采用 C 语言编程，实现啸叫频率和相应的功率放大器输出功率的计算算法，对测试过程中产生的啸叫进行实时监测并显示相应频率及输出功率。

啸叫频率检测通过比较器电路与单片机计数实现，具体方法为设置一定大小的阈值，作为判断啸叫信号的标准，比较器输出被测啸叫信号相同频率的 0V 到 5V 的方波。通过单片机测量方波的频率，方波的频率即为啸叫频率。

单片机采用基于 ARMV7-M 架构的 32 位微控制器 LM3S1138。芯片内部的通用定时器模块可配置成 16 位 PWM 模式，内建 8 路高速 ADC 输入通道，采样速率高达 1MHz，可配置成独立或差分输入模式；另外，该款芯片还有内置硬件看门狗定时器、64KB 单周期 FLASH、16KB 单周期 SRAM、输入捕捉、端口电平变化通知等功能，可以极大地简化外围电路的设计。

1.2.4 啸叫抑制模块方案选择与论证

(1) 方案一：调整距离法

调整距离法即改变话筒和扬声器的距离，是既避免啸叫又能提升扩音音量最有效的方法之一。话筒离扬声器越远，啸叫越小。

(2) 方案二：频率均衡法（宽带陷波法）

用频率均衡器补偿扩声曲线，把系统的频率响应调成近似的直线，使各频段的增益基本一致，提高系统的传声增益。实际上扩声系统在出现反馈自激时，其频率只是固定在某一点上的纯音，所以，只要用一个频带很窄的陷波器将此频率切除，即可抑制系统啸叫。

(3) 方案三：反馈抑制器法（窄带陷波法）

其原理也是通过陷波抑制啸叫的，跟踪反馈点频率，调整带宽，将声反馈消除。

(4) 方案四：反相抵消法

在音频放大电路中采用两个同规格的话筒分别拾取直达声和反射声，通过反相电路使反射声信号在进入功放前相位相互抵消，有效防止啸叫自激。

调整距离法是消极应对啸叫的方式，对提升扩音音量和防止啸叫的作用不大，故不可取；反馈抑制器法需要装置能自动跟踪反馈点频率并自动调节，技术含量较高，短时间内无法实现，舍去；反相抵消法需要两个同规格的话筒（麦克风），不符合竞赛要求；频率均衡法可利用啸叫检测模块检测到啸叫频率并将其切除，有效抑制啸叫。

综上所述，啸叫抑制模块选择方案二，通过频率均衡法实现啸叫有效抑制。

1.3 系统组成

综合以上讨论，确定系统设计的最终方案：对于前置放大的拾音电路，我们采用低噪的三极管 9014 对麦克风采集到的音频小信号进行放大；对于功率放大电路，我们基于功率放大器芯片 TPA3112D1 设计带有 LC 滤波的功率放大电路实现对信号的电压和电流放大；通过单片机 LM3S1138 和程序以及液晶显示屏搭建啸叫检测模块，实时监测啸叫并显示啸叫频率；采用频率均衡法，搭建均衡器电路，实现有效抑制啸叫功能。系统结构框图表示如下：

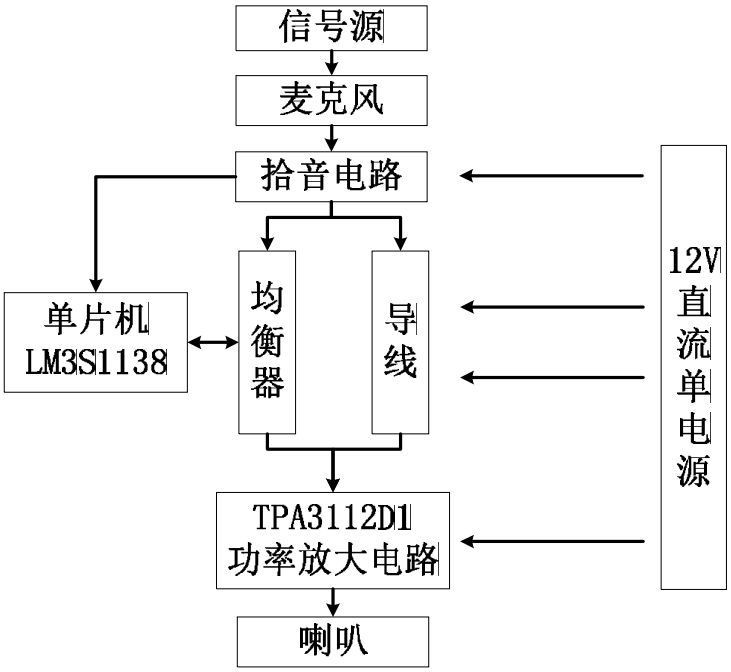


图 2　系统结构框图

2. 系统的电路与程序设计

2.1 主要模块的电路实现

2.1.1 拾音电路

利用低噪的三极管 9014 搭建拾音电路，对信号源产生的信号进行采集并放大。

电路原理图如右所示：

经仿真和实验测试，拾音电路的放大倍数可达到 80 倍，具体仿真波形可参见附录 2。

图 3　拾音电路图

2.1.2 功率放大电路

选择 12V 直流单电源供电电压，连接至 PVCC 引脚；音频输入电路信号为单端音源。功率放大电路的原理图如下：

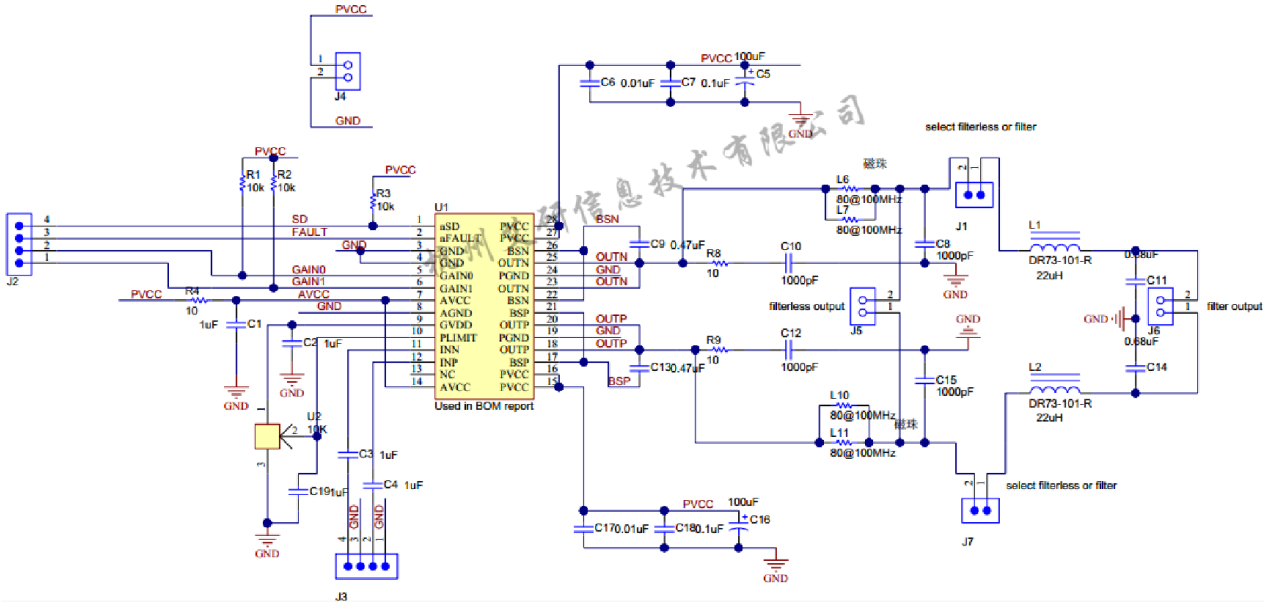


图 4　功率放大电路图

加入的 LC 滤波的原理图如下：

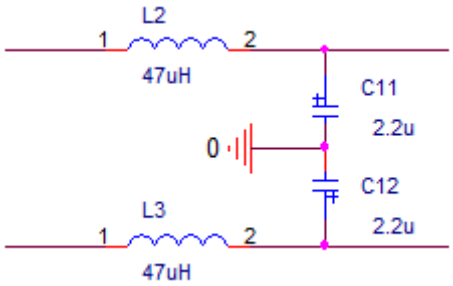


图 5　LC 滤波电路图

2.1.3 啸叫抑制电路

利用均衡器抑制啸叫的本质是对啸叫频率下的系统的频率特性进行补偿，需要补偿部分的频率可由啸叫检测电路检测得到。设计的均衡器是利用 5 个并联连接的 RLC 电路（由 RC 电路模拟电感电路），将补偿的中心频率分割为 5 份，分别进行控制，分割的频率可通过 $f_0 = \dfrac{1}{2\pi\sqrt{C_1C_2 \cdot R_1R_2}}$ 计算得到。

系统设计的分割频率为 189Hz、330Hz、1kHz、3.3kHz、10kHz，可控制啸叫频率范围为 200Hz ~ 10kHz 的电路。根据检测到的啸叫频率确定其相对应的频率区间，调节相应可变电阻器的滑触头，调整啸叫抑制电路参数，对系统电路实现在该频率下的特性补偿。

啸叫抑制电路的原理图如下，具体仿真波形可参见附录 2：

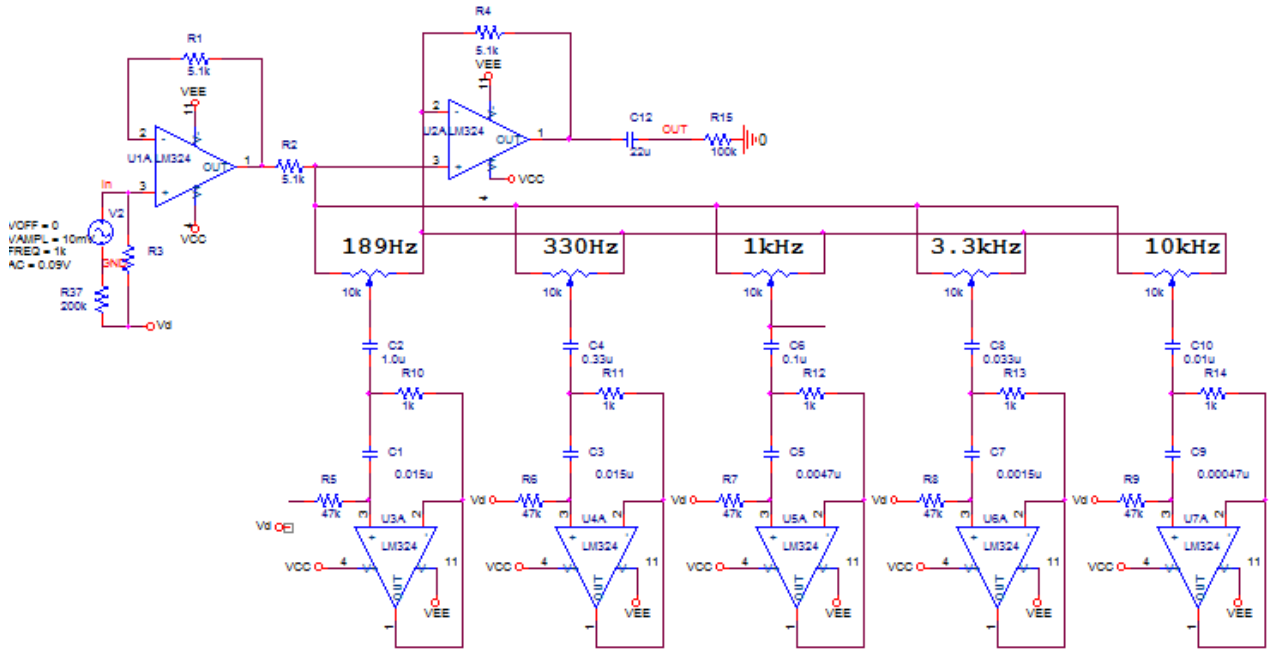


图 6　啸叫抑制电路图

2.2 程序设计

系统软件设计采用 C 语言编程，便于实现啸叫频率的实时监测和显示。程序实现思路如下：将 Timer0 配置成一个 32 位周期定时器，定时时长为 1s。将 Timer2A 配置成 16 位边沿捕获计数器，通过对 Timer0 定时的 1s 内方波的下降沿数进行计数来获得方波的频率，即获得啸叫频率。

3. 系统测试与结果分析

3.1 测试仪器

表 3.1.1　测试用仪器与设备

序号	名称	型号及规格	数量	备注
1	数字万用表	VC9801A	1	
2	示波器	GDS-1152A-U	1	
3	信号源	AFG-2225	1	产生 20mV 输入信号
4	电源	JC3003A-3	1	12V 直流单电源供电

3.2 指标测试与分析

表 3.2.1　基本音频功率放大器

测试项目	最大不失真功率	功率范围	频率相应范围	电路整体效率	电脑音源
理论结果	5W	50mW~5W	200Hz~10kHz	>=80%	输出信号清晰
测试结果	4.8W	45mW~4.8W	330Hz~8kHz	67%	输出信号清晰
误差	4%				

表 3.2.2　改进音频功率放大器

测试项目	调整输出功率	频率计算及显示	功率显示	有效抑制啸叫	发挥部分
理论结果	可实现	可实现	可实现	可实现	
测试结果	可实现	可实现	不可实现	可实现	

3.3 结果分析

从系统调试和测试结果可以看出：音频功率放大器基本上能完成系统设计的基本要求和改进部分，但系统的基本性能劣于设计要求。

啸叫频率随测试的信号源的改变变化较大，且非常容易受到外界环境干扰源的影响，因此啸叫检测频率的算法可以进一步改进。

4．结论

采用低噪的三极管 9014 搭建的拾音电路和基于功率放大器芯片 TPA3112D1 搭建的带有 LC 滤波的功率放大电路相结合，可以实现一个基本音频功率放大器的音频功放功能，能完成对台式麦克风音频信号的放大，并送喇叭输出。

将由均衡器构成的啸叫抑制电路加入基本音频功率放大器中，能有效抑制啸叫，并正常播放音频信号；利用单片机 LM3S1138 实现的啸叫检测模块可以检测并显示啸叫频率。在基本音频功放电路的基础上增加的啸叫检测和啸叫抑制功能，完善了音频功率放大器。

综上所述，本系统基本满足设计要求。

参考文献

[1]　AY-TPA3112D1 EVM 用户指南. http://www.hpati.com.

附录

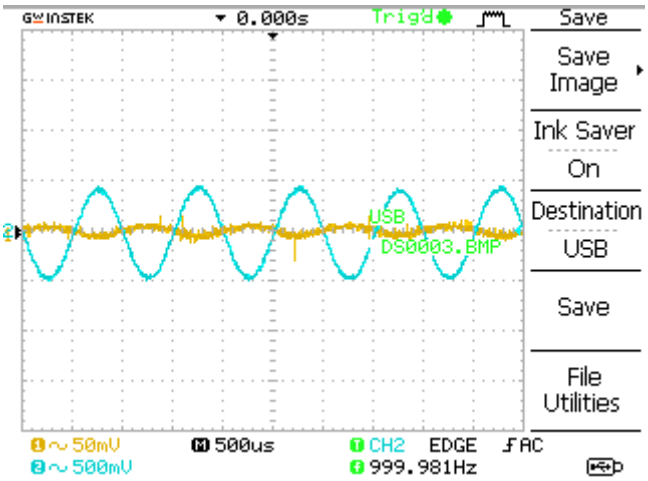
附录 1　主要元器件清单

附表 1.1　主要元器件清单

序号	名称	型号及规格	数量	备注
1	功率放大器芯片	TPA3112D1	1	
2	运算放大器芯片	LM324		
3	三极管	9013、9014	3	
4	喇叭	5W　8Ω	1	
5	单片机	LM3S1138	1	
6	液晶显示 LCD	1602	1	

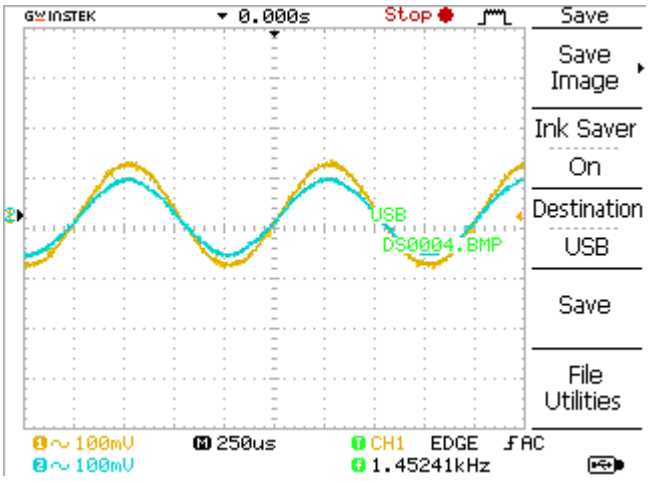
附录 2　电路仿真

拾音电路实现信号采集及放大：

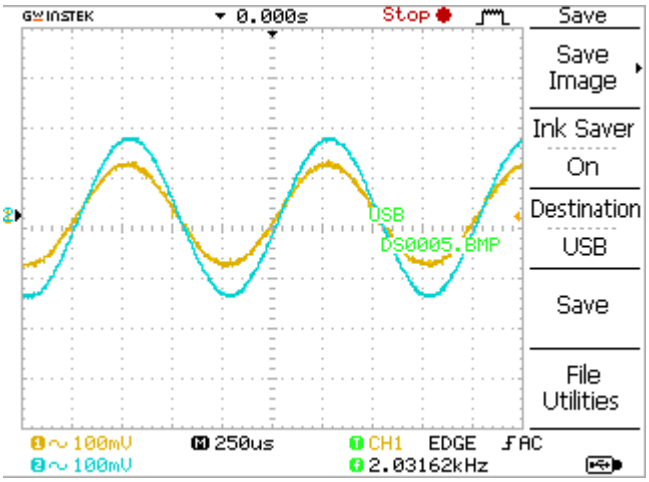


附图 2.1　拾音电路仿真

啸叫抑制电路通过补偿系统因啸叫改变的频率特性实现啸叫的有效抑制。输出信号的幅值随可变电阻器滑触头的改变发生如下所示的变化：

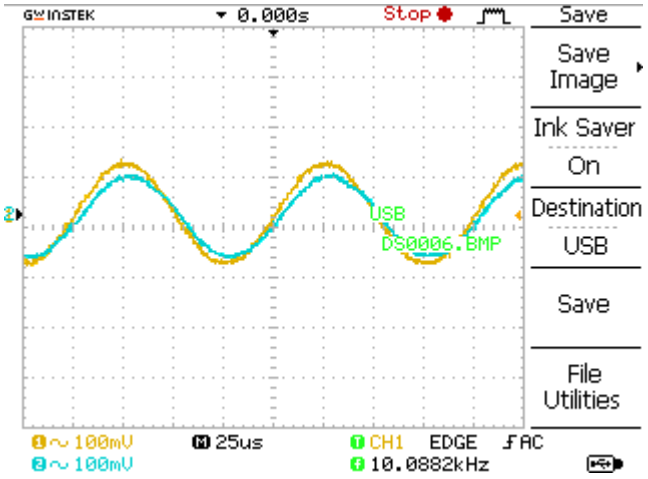


附图 2.2.1　啸叫抑制电路仿真



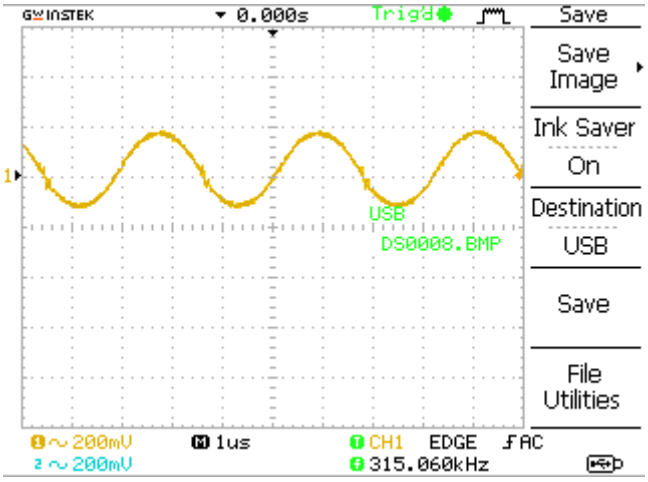
附图 2.2.2　啸叫抑制电路仿真

输出信号的相位也随可变电阻器滑触头的改变发生如下所示的变化：



附图 2.2.3　啸叫抑制电路仿真

功率放大电路实现电流和电压的放大：



附图 2.3　功放电路仿真

附录 3　程序清单

main.c

```
#include   "systemInit.h"
#include   <timer.h>
#include   "lcd12864.h"
#include   <stdio.h>

#define    CCP4_PERIPH       SYSCTL_PERIPH_GPIOF
#define    CCP4_PORT         GPIO_PORTF_BASE
#define    CCP4_PIN          GPIO_PIN_7

//  定时器及计数器初始化
void timer_counter_init(void)
```

```
{
  //  配置 Timer0 为 32 位周期定时器，定时周期为 1s
  SysCtlPeriEnable(SYSCTL_PERIPH_TIMER0);                    //  使能 Timer0 模块
  TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);         //  配置 Timer0 为 32 位周期定时器
  TimerControlStall(TIMER0_BASE,TIMER_A,true);               //  Timer0 在单步调试下暂停运行
  TimerLoadSet(TIMER0_BASE, TIMER_A, 6000000UL);             //  设置 Timer0 初值，定时 1s

  TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);           //  使能 Timer0 超时中断
  IntEnable(INT_TIMER0A);                                    //  使能 Timer0 中断

  //  配置 Timer2A 为 16 位输入边沿计数捕获计数器
  SysCtlPeriEnable(SYSCTL_PERIPH_TIMER2);                    //  使能 Timer2A 模块
  SysCtlPeriEnable(CCP4_PERIPH);                             //  使能 CCP4 所在的 GPIO 端口(PF)
  GPIOPinTypeTimer(CCP4_PORT, CCP4_PIN);                     //  配置 CCP4 管脚(PF7)为脉冲输入

  TimerConfigure(TIMER2_BASE, TIMER_CFG_16_BIT_PAIR |        //  配置 Timer2A 为 16 位事件计数
器
                                TIMER_CFG_A_CAP_COUNT);

  TimerControlEvent(TIMER2_BASE,                             //  控制 Timer2A 捕获 CCP 负边沿
                    TIMER_A,
                    TIMER_EVENT_NEG_EDGE);

  TimerLoadSet(TIMER2_BASE, TIMER_A, 40000);                 //  设置计数器初值
  TimerMatchSet(TIMER2_BASE, TIMER_A, 10000);                //  设置事件计数匹配值

  TimerIntEnable(TIMER2_BASE, TIMER_CAPA_MATCH);             //  使能 Timer2A 捕获匹配中断
  IntEnable(INT_TIMER2A);                                    //  使能 Timer2A 中断
  TimerEnable(TIMER2_BASE, TIMER_A);                         //  使能 Timer2A 计数

  IntMasterEnable();                                         //  使能处理器中断
}

//  主函数（程序入口）
int main(void)
{
    jtagWait();                                              //  防止 JTAG 失效，重要！
    clockInit();                                             //  时钟初始化：晶振，6MHz
    timer_counter_init();                                    //  定时器及计数器初始化
    softspi_init();                                          //  lcd 显示相关 IO 的初始化
    lcd_init();                                              //  lcd 初始化

    TimerEnable(TIMER0_BASE, TIMER_A);                       //  使能 Timer0 计数
```

```
    for (;;)
    {
    }
}


//  定时器的中断服务函数
void Timer0A_ISR(void)
{
    unsigned long ulStatus;
    unsigned long countervalue;
    unsigned long frequency;
    char c[20];

    ulStatus = TimerIntStatus(TIMER0_BASE, true);              //  读取中断状态
    TimerIntClear(TIMER0_BASE, ulStatus);                      //  清除中断状态，重要！

    if (ulStatus & TIMER_TIMA_TIMEOUT)                         //  如果是 Timer0 超时中断
    {
      countervalue = TimerValueGet(TIMER2_BASE,TIMER_A);       //  得到计数器的当前值
      TimerLoadSet(TIMER2_BASE, TIMER_A, 40000);               //  设置计数器初值
      frequency = 40000 - countervalue;                        //  计算方波频率

      sprintf(c, "频率:%ldHz", frequency);
      lcd_write(com,0x01);                                     //  清屏，将 DDRAM 的地址计数器归零
      Show(0x90,c);
    }
}

//  计数器的中断函数
void Timer2A_ISR(void)
{
    unsigned long ulStatus;

    ulStatus = TimerIntStatus(TIMER2_BASE, true);              //  读取当前中断状态
    TimerIntClear(TIMER2_BASE, ulStatus);                      //  清除中断状态，重要！

    if (ulStatus & TIMER_CAPA_MATCH)                           //  若是 Timer2A 捕获匹配中断
    {
        ;
    }
}
```