语音信号采集器

基于 Arm Cortex-M4 的 MCU 音频处理系统

张维通，153402010318；张泽祥，153402010414；雷光荣，153402010415

第一部分　设计概述

1.1 设计目的

在大数据以及人工智能的浪潮下，语音信号的处理及合成变成了尤为重要的一个部分，同时 Arm Cortex-M4 处理器使得利用显著低廉且简单的微控制器进行实时音频处理成为可能。本次设计是基于 Arm Cortex-M4 内核的 MCU 音频处理系统，对语音信号实时采集并进行 DSP 处理。对语音信号进行消除环境噪音，并通过 QN8027 调频发射。组成了一个无线麦克系统，实现了音频处理在实际情景下的使用。实现了很宽的动态范围，低失真，高信噪比的语音信号采集及其处理系统。

1.2 应用领域

高速数据采集技术已经广泛的应用在雷达、导弹、通信、声呐、遥感。而在近年来比较火热的大数据处理以及人工智能的发展，音频信号的采集处理成为了目前发展较为迅速的研究课题之一。ARM 嵌入式微处理器，以其完整的体系结构发展系列，极小的体积、极低的功耗、极高的性能，在众多嵌入式微处理器中脱颖而出，从而使得应用领域更加广泛，智能仪器，生物医学等。本次设计的音频处理系统结合调频发射实现了在短距离的无线麦克音频通信。

1.3 主要技术特点

本次设计将 Arm Cortex-M4 内核以及外部 QN8027 调频发射模块相结合。故其主要性能指标如下：

（1）Fs=25K;

（2）16 位 ADC 采集数据；

（3）预加窗；

（4）帧长 256（4*256 us 在 10~30ms 可认定语音信号特征基本不变）

（5）16 位 DAC 输出（MAX541）;

（6）3500Hz 椭圆低通滤波（MAX297）;

（7）载波频率范围 88MHz~108MHz;

（8）最大频偏 75Hz;

（9）频道间隔 200KHz;

（10）最大音频功率输出 0.5W;

（11）通信距离在 10~20 米;

Fs=25K 的采样速率已经达到的 FM 的指标，同时 16 位的高速率采集使得在量化噪声的处理方面更加的出色。在外部的硬件处理方面，器件的选择都是高性能的器件，使得将外部的可避免噪声降到最低。调频发射模块的选择也是采用了集成的调频发射芯片 QN8027。

1.4 关键性能指标

在设计完成后，测试过程中不断调试以达到最佳处理效果。以下是最后调试后的关键性能指标：

（1）量化信噪比 SNR= 6.02B-7.2;

（2）Fs=25K,预滤波（DSP 为中间件）;

（3）两路音频采集;

（4）自适应滤波;

（5）基于 PID-PDB-ADC-DMA-CPU 的处理;

（6）音频信号在接收端失真度较低，其噪音处理较好。

利用 CMSIS-DSP 等运算中间件，选择自适应滤波，并通过两个麦克风采集的信号，减小音频中的环境噪声。在实际测量值观察波形和试听实际音质，以上指标全部达到。在收音机接收端接收到的音频噪声和啸叫影响较低，效果良好！

第二部分 系统组成及功能说明

2.1 整体介绍

无线麦克系统，该系统主要由麦克风及信号放大模块、AD 数据采集模块、DSP 处理模块（带通滤波，自适应滤波）、DAC 数据转化模块、QN8027

调频发射模块。主要构成如下图所示。

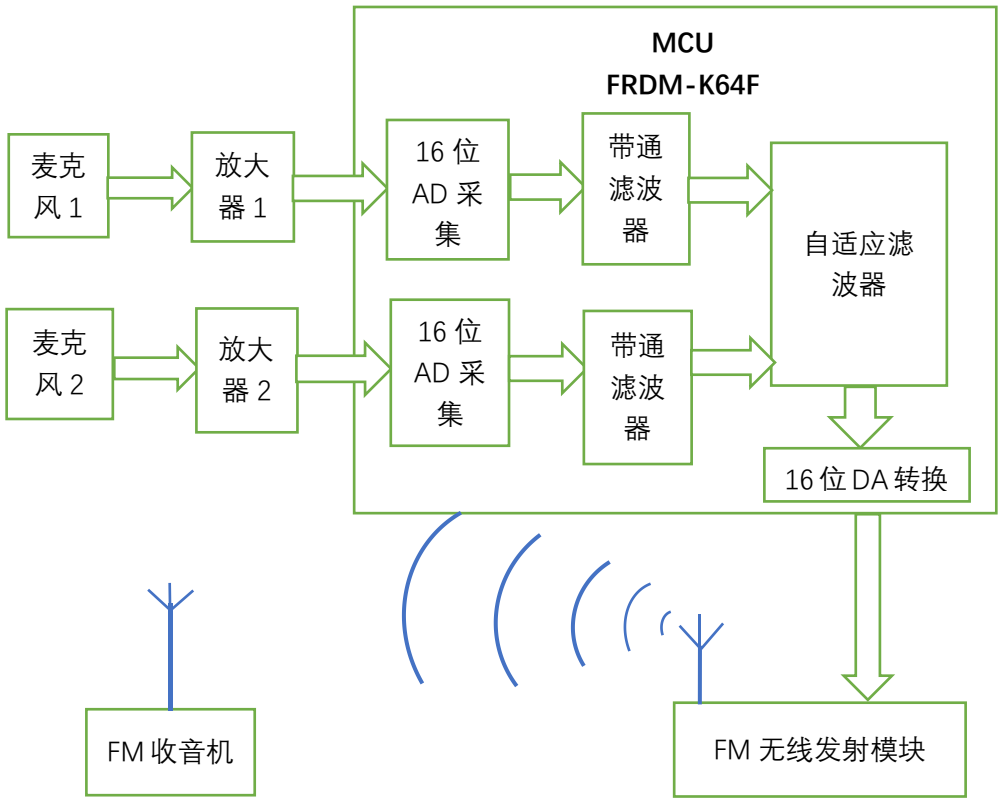


图 1　整体结构框图

麦克风及信号放大模块是对音频信号的收集和转化，为了后续的数据采集处理，需要对收集的信号进行放大，方便 AD 的采集。在 AD 采集模块是使用 Arm Cortex-M4 内核的 AD 模块，将语音信号离散化处理，在通过分帧的方法一帧一帧的进行处理，在 DSP 处理模块是通过软件程序编写控制，首先是一个带通滤波器将声音频率范围外的噪声全部滤除，再通过自适应滤波器将两路采集的音频信号处理，抵消噪声。将处理后的信号通过 DAC 转化模块将信号还原为模拟音频信号输出。在调频发射模块，将处理后的音频信号，调频发射。调频发射频段可调，步进 200KHz。在接收机处接受语音信号。以上是各个模块的作用介绍和模块间的信号传递和处理。

2.2 各模块介绍

2.2.1 麦克风及信号放大模块

语言信号的输入是通过小麦克风，将声音信号转化为电信号，正常的无源电路驱动麦克风，出现功率不足的情况，放大效果不好，噪音大。在首先通过前置三级管放大电路将音频信号一级放大，再通过音频放大运放 386

进行二级放大，这样在保证音频信号不失真的情况下，可以保证最大输出。下图 2 是该部分的电路设计。

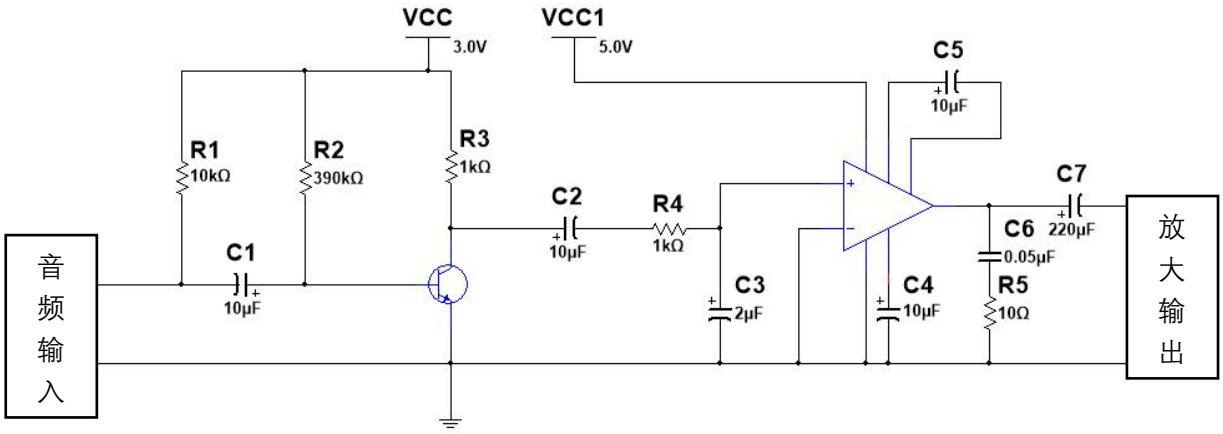


图 2 音频放大电路

该电路放大倍数的计算分为两个部分，首先是三级管放大电路的计算，其中三级管使用 9013，其β=200，已知三级管的$U_{CES}=0.7V$，且供电电压为 3V，图中电容C_2用于连接放大电路与负载，在电子线路中其连接作用的电容为耦合电容，利用电容连接电路称为阻容耦合，通过电路可以求出静态工作点。

$$I_{BQ}=\frac{VCC-U_{BEQ}}{R_B} \qquad ①$$

$$I_{CQ}=\beta I_{BQ} \qquad ②$$

$$U_{CEQ}=VCC-I_{CQ}R_E \qquad ③$$

通过以上①②③公式可以计算出其静态工作点为 1.8V，处于三级管的放大区，故电路处正常放大。

第二级放大电路是用到了可单电源供电的功放 LM386，该运放是一种音频集成功放，具有自身功耗低、更新内链增益可调整、电源电压范围大、外接元件少和总谐波失真小等优点的功率放大器，广泛应用于录音机和收音机之中。由于在外部 1、8 引脚内部有电容，则固定有 20 倍的增益放大，然而 20 倍的增益放大不够，则可以在 1、8 引脚之间加10 μF 的电容，这样可以达到 200 的增益放大。使得音频信号足够发射，而不引起失真。该部分电路在 MULTISIM 仿真软件中，效果正常。

以上便是该音频放大电路的设计过程。

2.2.2 AD 数据采集模块

该模块是由程序配置工作，下图 3 是该部分的流程图。

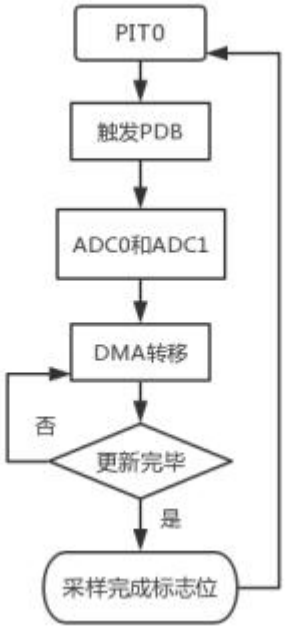


图 3 流程图

在处理过程中，PIT0 来触发 PDB，同时 PDB 可以控制两路 AD 的工作，即 ADC0 和 ADC1。传递至 DMA 后进行判断每一帧的处理结果进行判断。以上是 ADC 的处理模块。详细程序见附录。

2.2.3 DSP 处理模块（带通滤波，自适应滤波）

(1) 数字带通滤波器的设计

数字带通滤波器的设计是结合 MATLAB 来产生传递函数的参数。MATLAB 提供一款滤波器的设计和分析工具 FDATOOL，可以简化 IIR 滤波器的设计系数计算。系数直接 II 型二阶部件的格式从 MATLAB 工作空间中导出。

为了实现 FDATOOL 设计的滤波器，需要按照下列步骤执行

第一步:首先单击 FDATOOL 界面左下方的滤波器设计(Design fite)按键图标，进入滤波器设计界面。

第二步:单击 FDATOOL 界面上方工具栏上的滤波器参数设置(Filter Specifications)工具，进入滤波器参数设置界面。

第三步:单击滤波器响应类型(Response Type)部分的带通滤波器(Bandpass)选项，指定设计带通滤波器。

第四步:在设计方法(Design Method)部分的 FIR 下拉列表框中，选择等纹波(Equiripple)设计方法。需要注意的是，除了凯塞窗可以通过调整 β 参数来改变滤波器阻带衰减外， 其他窗函数的阻带衰减是无法调整的，增加滤波器阶数只能改变滤波器的过渡带性能。最优滤波器设计方法(等纹波滤波器)可通过增加滤波器阶数改善阻带衰减性能，

第五步: 根据设计要求，设置滤波器截止频率。

第六步:设置滤波器阶数(Specify order)参数后，单击 FDATOOL 界面下方的“DesionFilter"按键开始滤波器设计。

第七步:观察 FDATOOL 中的频率响应，调整滤波器阶数，直到满足设计要求。

至此，我们使用 FDATOOL 完成了带波滤波器的设计，用户可以通过单击“Analysis -FilterCefiets"菜单来查看滤波器系数，或通过单击“angeis XLIX Cficient C.COE)IFile"菜单直接生成 FPGA 所需的滤波器系数配置文件。下图 4 是 MATLAB 中 FDATOOL 的设计页面参数配置。

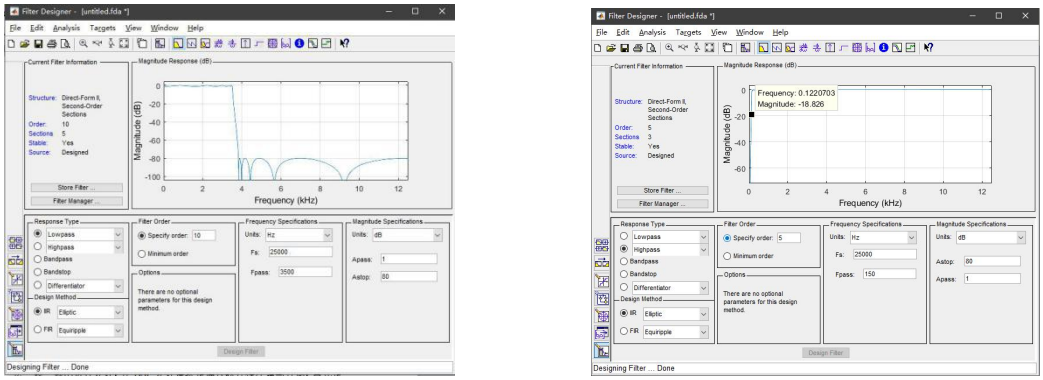


图 4 FDATOOL 的设计页面参数配置

(2) 自适应滤波

自适应滤波器用于所涉及信号的特征或者统计特性要么未知、要么时变的情形。通常，非自适应的 FTR 或者 IR 滤波器都是参照特定的信号特征进行设计的。但是，如果滤波器遇到的信号特征与设计此滤波器时所参照的典型特征不符，则此滤波器的性能不会是最优的。而自适应滤波器的系数可以自行调节，使得滤波器的性能随时间得到某种程度的提高，最终达到最优。因此，当信号

特征并不确定或者当信号特征是时变的时候，都要用到自适应滤波器。

自适应系统拥有超过非自适应系统的潜力。然而，由其定义可知，自适应系统是非线性的，其分析难度高于线性时不变系统。不过，本章关注的是线性自适应系统，也就是说，当适应性受到抑制时，系统将拥有线性特征。该结构的部件包括:一个可调节滤波器，套性能测量装置 (此处是一个比较器，用来测量自活应滤波器的输出与期望输出之间的瞬时误差)以及一套自适应装置或者算法。 下图 5 是自适应滤波器的原理框图。

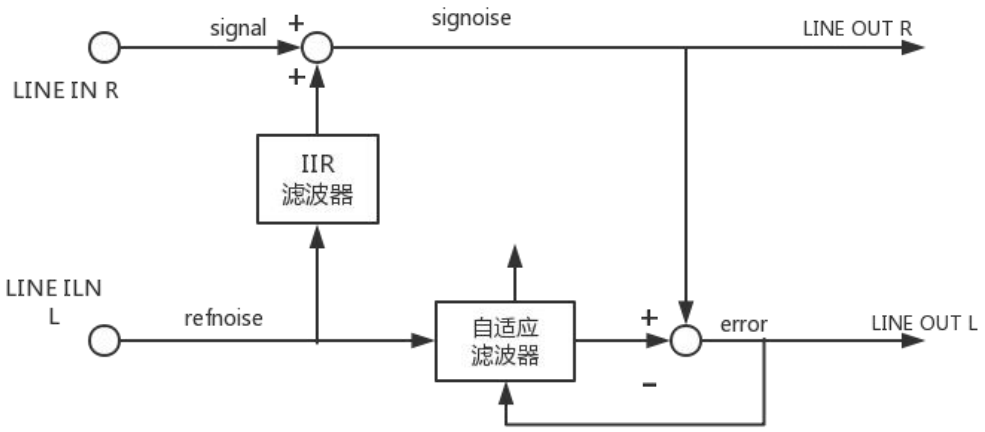


图 5 自适应滤波器的原理框图

2.2.4 QN8027 调频发射模块

（1）硬件电路的设计

在该模块电路设计当中，采用 QN8027 调频发射模块。QN8025 是一个高度数字化的低中频芯片，允许外部信号进行调整。 接收射频信号首先由一个低噪声放大器，然后向下转换为一个中间频率传给正交混频器。为了改善音质，抑制噪音，正交混频器可被编程在高端或低端注入。每个通道使用滤波器抑制干扰信号。 它还可以接收 RDS 信息，内容可以通过 LCD 显示。高功率 32Ω负 载音频输出，直接耳机驳接，无需外接音频放大。 调频发射机:与 QN8025 接收机模块配对的是以 QN8027 芯片为中心的发射模块，可以实现音频和 RDS 的发射。

QN8027FM 高性能发射芯片，尺寸小，外围电路也只有两个电阻电容。立体声信噪比 33dB，单声道信噪比 73dB，谐波失真度低。发射方面可以支持全频段

76-108MHZ 发射，发射距离可达 50M，发射频率可调。与题目要求相比较，QN8027 完全满足设计要求。其基于 QN8027 的发射机电路图如图 6 所示。

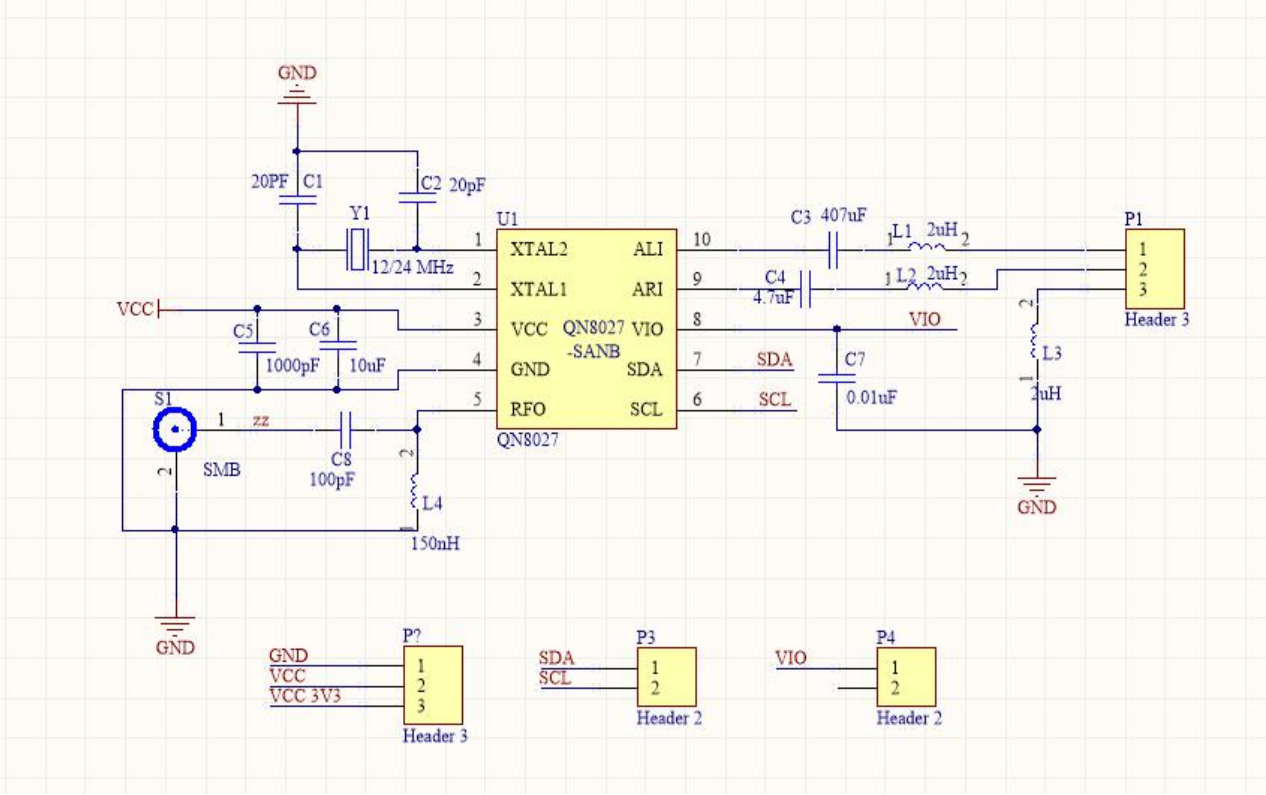


图 6 QN8027 的发射机电路

为了保证音频调频发射功率问题，也可以在发射前，加一级功放电路，其电路如图 7 所示。

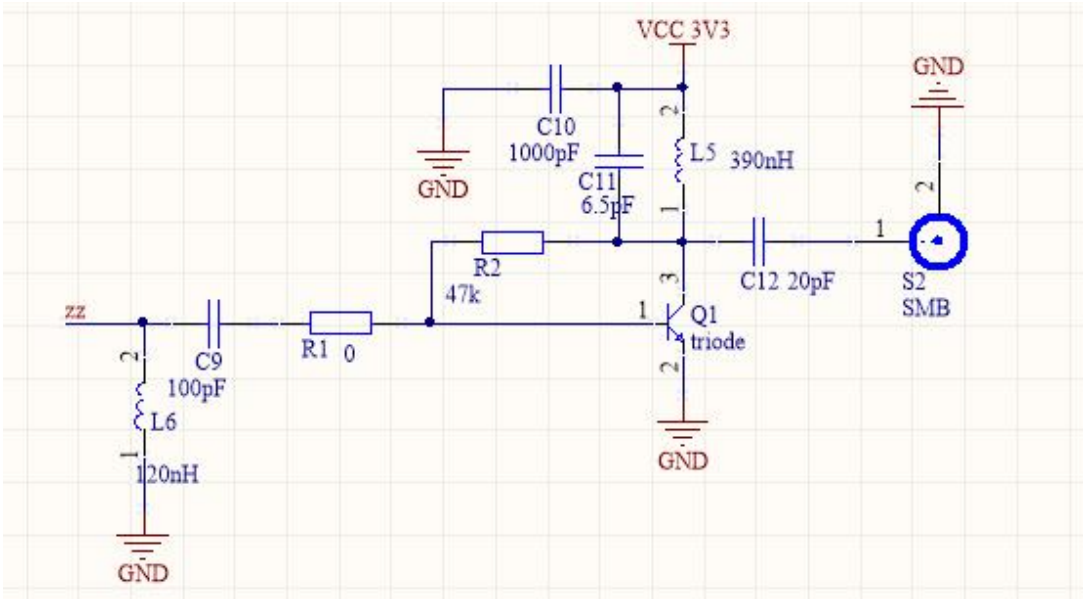
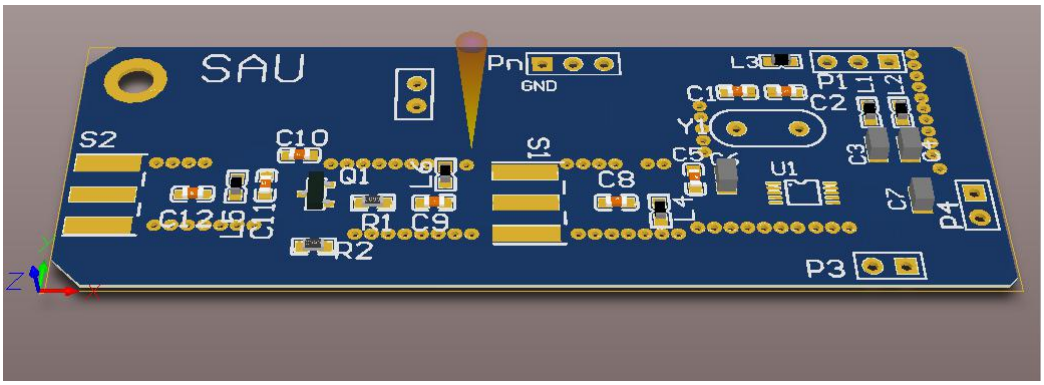


图 7 音频调频发射功放电路

通过 Altium Disigner 软件绘制 PCB 版图，下图 8 为该模块的 PCB 版图。



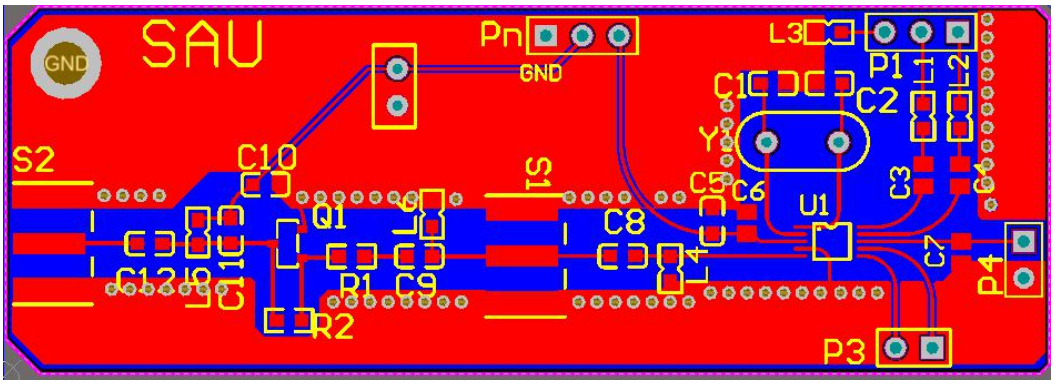


图 8 模块的 PCB 版图

（2）程序配置部分

在该系统中，单片机对接受发送模块的控制，主要是 IIC 总线的读写，而程序的设计分为接受模块可发送模块的程序编程。由于需要对 200K 频率的步进调试，则需要添加按键来时实现对载频的 200K 步进，同时加入 LOED 来显示发射的载频，这样更加人性化，于此同时，需要 IIC 总线来控制读写及初始化。下图 9 是总体的程序流程图。

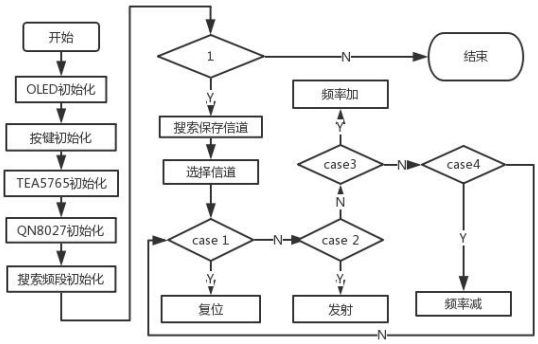


图 9 程序流程图

下面分别介绍发射模块的程序编写。

在该模块的编写时，首先是 IIC 的数据总线控制，来实现对发射模块 QN8027D 的控制。下图 10 是 QN8027 的时序图。

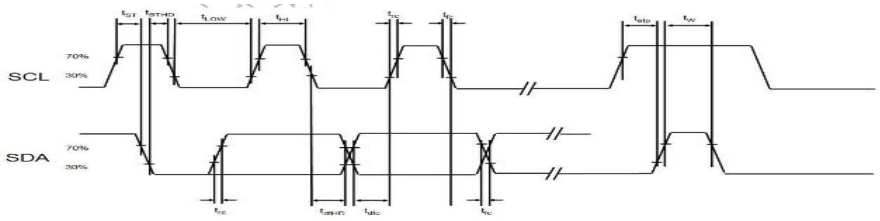


图 10　QN8027 的时序图

根据上图时序，分析可知其芯片程序操作如下：

1.初始化

（1）将寄存器 0x00 设置为 0x81　//对所有寄存器复位

（2）延迟 20ms　//等待复位完成

（3）将寄存器 0x18 设置为 0x40　//选择 32.768KHz 外部时钟输入,这一行与客户硬件设计有很大关系

（4）将寄存器 0x2F 设置为 0x90.

（5）将寄存器 0x40 设置为 0x01.

（6）将寄存器 0x46 设置为 0x05.

（7）将寄存器 0x47 设置为 0xD0.

（8）将寄存器 0x48 设置为 0xA8　//设置第 7 脚 RFI 高阻抗输入

（9）将寄存器 0x45 设置为 0x39.

（10）将寄存器 0x4A 设置为 0x08.

（11）将寄存器 0x00 设置为 0x51　//对有限状态机作校验

（12）将寄存器 0x00 设置为 0x01　//对有限状态机作校验

（13）延时 200ms.　//等待有限状态机校验完成

2.工作模式的设置

（1）设置芯片工作模式为 standby（寄存器 0x00=0x21），在 SDK 软件中称为 sleep 模式。

（2）设置芯片工作模式为 idle（寄存器 0x00=0x01）,在 SDK 软件中称为 wakeup 模式。

（3）设置芯片工作模式为 RX（寄存器 0x00=0x11）,即 FM 手动接收模式。

此项操作需要调用的函数： QND_SetSysMode(UINT16 mode) // 设置工作模式 。通过对 QN8027 的设置，这样就可以正常工作。

第三部分 完成情况及性能参数

由于是音频信号的处理在，音质在报告中难以展示，以下部分音频以示波器图像展示。

首先是音频收集和信号放大电路的测试，测试得三级管的静态工作点如下表。

器件	集极	集电极	发射极
三级管 9013	0.7V	2.1V	0V

分析三级管的管脚电压可知，三级管工作在放大区。同时观察音频信号在确定放大电路正常工作后，测试各个点的波形分析如下。

下图 11 是语音采集的波形。分别是音乐输入和人声输入。

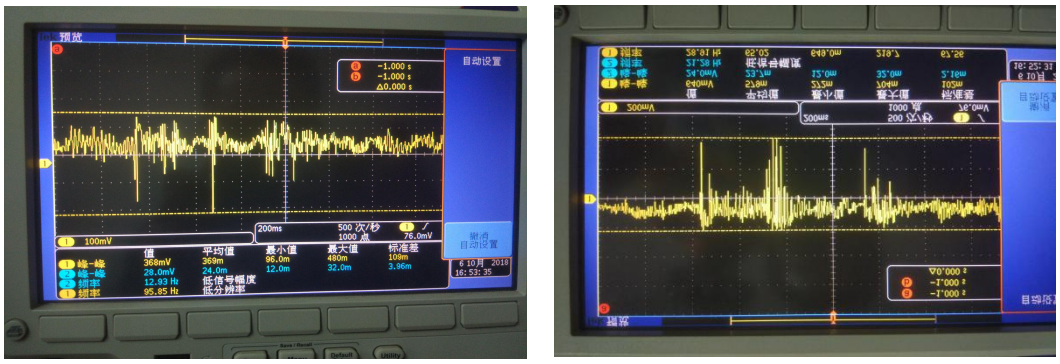


图 11 语音采集的波形

下图 12 是处理前后的语音波形比较。

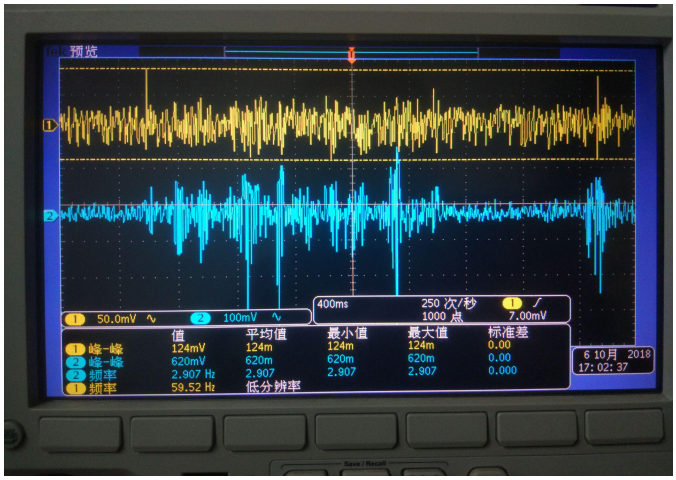


图 12 处理前后的语音波形比较

下图 13 是音频信号经过调频发射模块后，产生的调频波。



图 13 调频发射模块产生的调频波

以上是示波器观察到的音频处理后的变化。效果比较明显，噪声大幅度的降低。在视屏中将详细的展示音频处理后的效果。

第四部分 总结

4.1 主要创新点

本次设计的基于 Arm Cortex-M4 的 MCU 音频处理系统为主得无线麦克系统，利用了 K60 开发板和 QN8027 调频发射模块。在音频 DSP 处理时使用了 Arm Cortex-M4 的内核 DSP 函数，主要是创新的是在没有使用声卡的情况下处理音频信号，且处理效果良好，噪音干扰较小。结合调频发射模块，实现无线麦克的扩音，组成一个实用的调频通信系统。

4.2 可扩展之处

本次设计的是音频信号的采集系统，扩展之处在与音频采集处理后的部分扩展性好，该系统无线麦克扩音就是典型的扩展应用。同时处理后的音频可以进行压缩处理以及其他的扩展应用。也可以降低话筒回声，制造音频特效。

4.3 心得体会

随着国庆假期的的结束，我们的嵌入式设计大赛也随之接近尾声。回想当时，我和队友三人在学期末尾一边复习考试，一边谋划即将到来的大学生电子设计大赛。突然看到有同学

发过来的关于嵌入式芯片与系统竞赛的相关通知，由于这是一次难得的机会，加之结束时间是在下学期，眼看报名截止时间要到了，于是我们抱着试一试的希望冲忙填写信息递交了报名申请。就在我们都没有注意的时候，有一天收到了申请通过的消息。

就在紧张的期末考试结束之后，我们便开始了本次竞赛的准备工作。在这期

间，每个人都很认真地做着自己的任务，基本上是一有空就去实验室商讨对策，实施设计。没过多久，我们便收到了来自大赛组的开发板，是一块 M4 内核的 K64 开发板，结构紧凑，做工精致，还附带网络接口。在看了大赛组提供的参考资料之后，我们决定做一个队语音信号进行采集降噪的嵌入式小系统。该系统的主要部分便是队信号进行采集后降噪处理的硬件电路，以及各种滤波器算法。

在这期间，我们经历了电子设计大赛的紧张与疲惫，专业实习的忙碌，以及招聘会的四处奔波，搞得我们疲惫不堪。但是，我们始终没有放弃，决定一直做到比赛截止。就在这个国庆假期，当很多人都在享受假期旅游时，我们依然坚持实验室、寝室、食堂三点一线的生活，在这期间我们还通宵做过，晚上睡实验室杯寒冷的北方天气给冻醒过。经过许久的坚持努力，设计作品基本完成了。虽然它有很多不足与瑕疵，有很多待改进之处，但是我们做过、努力过、经历过，浙江在我们的人生经历中成为一段不可磨灭的回忆。当然，通过本次竞赛，我们也学到了很多平时上课所学不到的东西，它将让我们离成功更近一部。

第五部分 参考文献

［1］阎石主编．数字电子技术[M]. 北京：高等教育出版社，2006 年

［2］陈振官等编著．新颖高效声光报警器[M]. 北京：国防工业出版社，2005 年

［3］DonaldS.READ 编著 李磊 译．基于 ARM Cortex-M4 的 DSP 系统的开发[M]. 北京：机械设计与制造出版社，2010 年

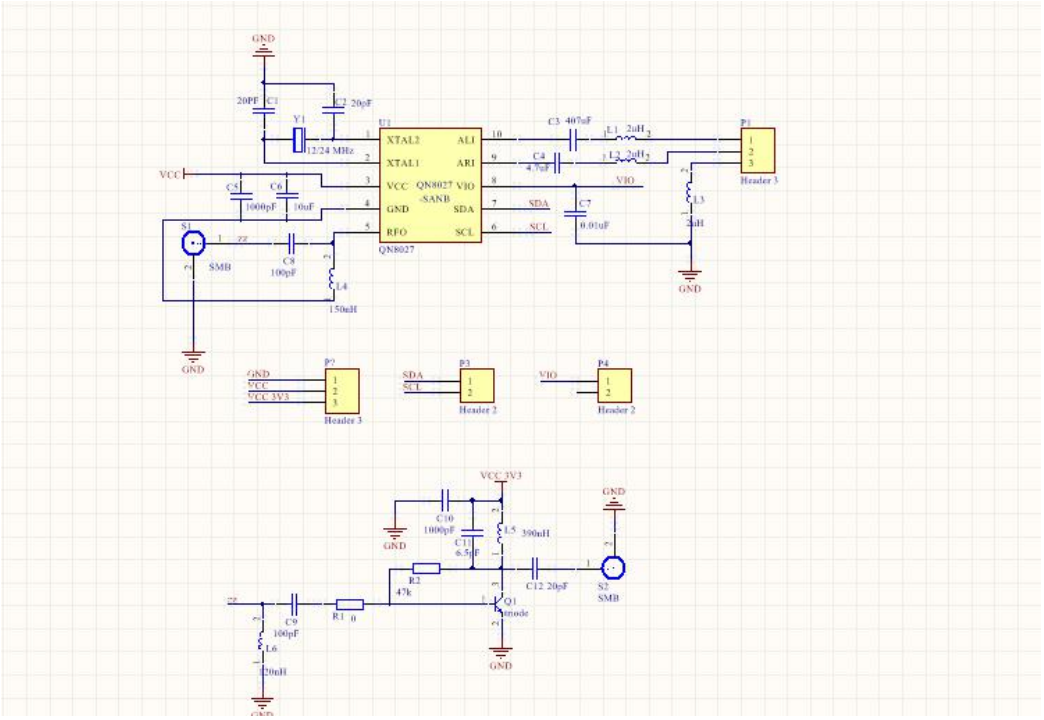
[4]肖广兵等编著． 嵌入式开发实例[M]. 电子工业出版社社，2013 年

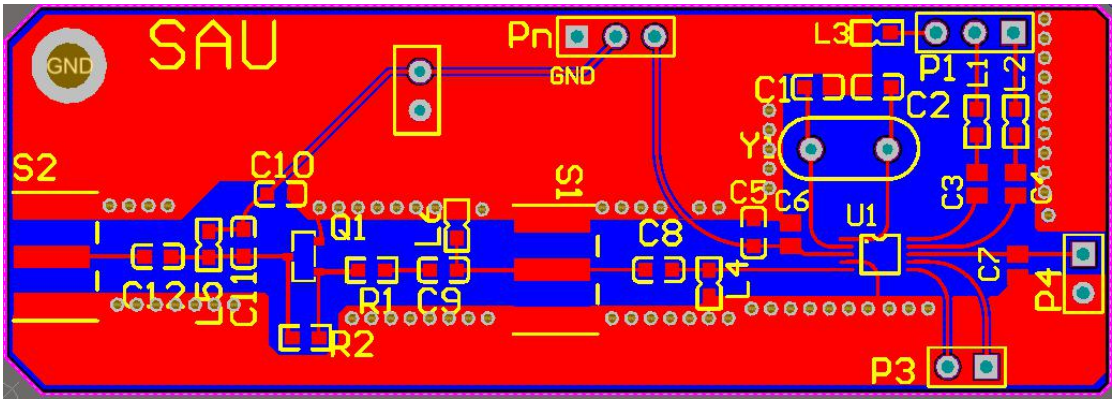
[5]张磊等编著. 语音信号处理[M]. 清华大学出版社. 2013 年

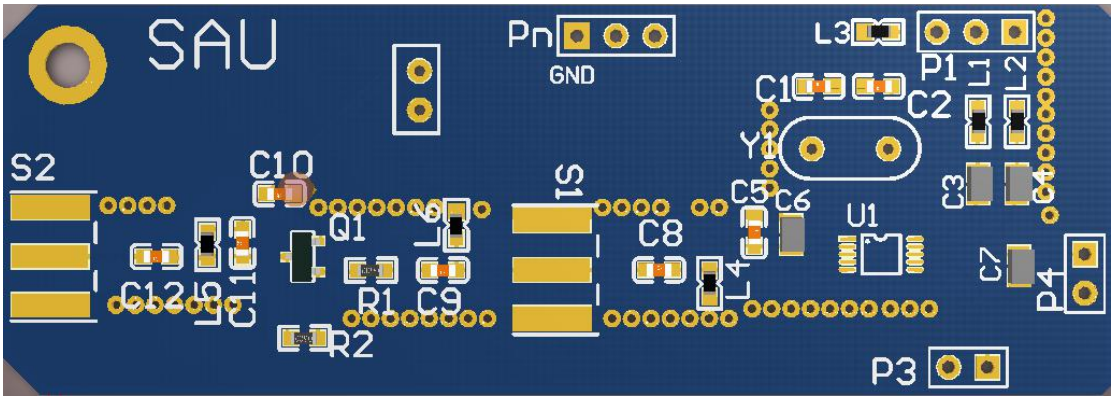
[6]赵力主编. 语音信号处理 [M]. 机械工业出版社. 2009 年

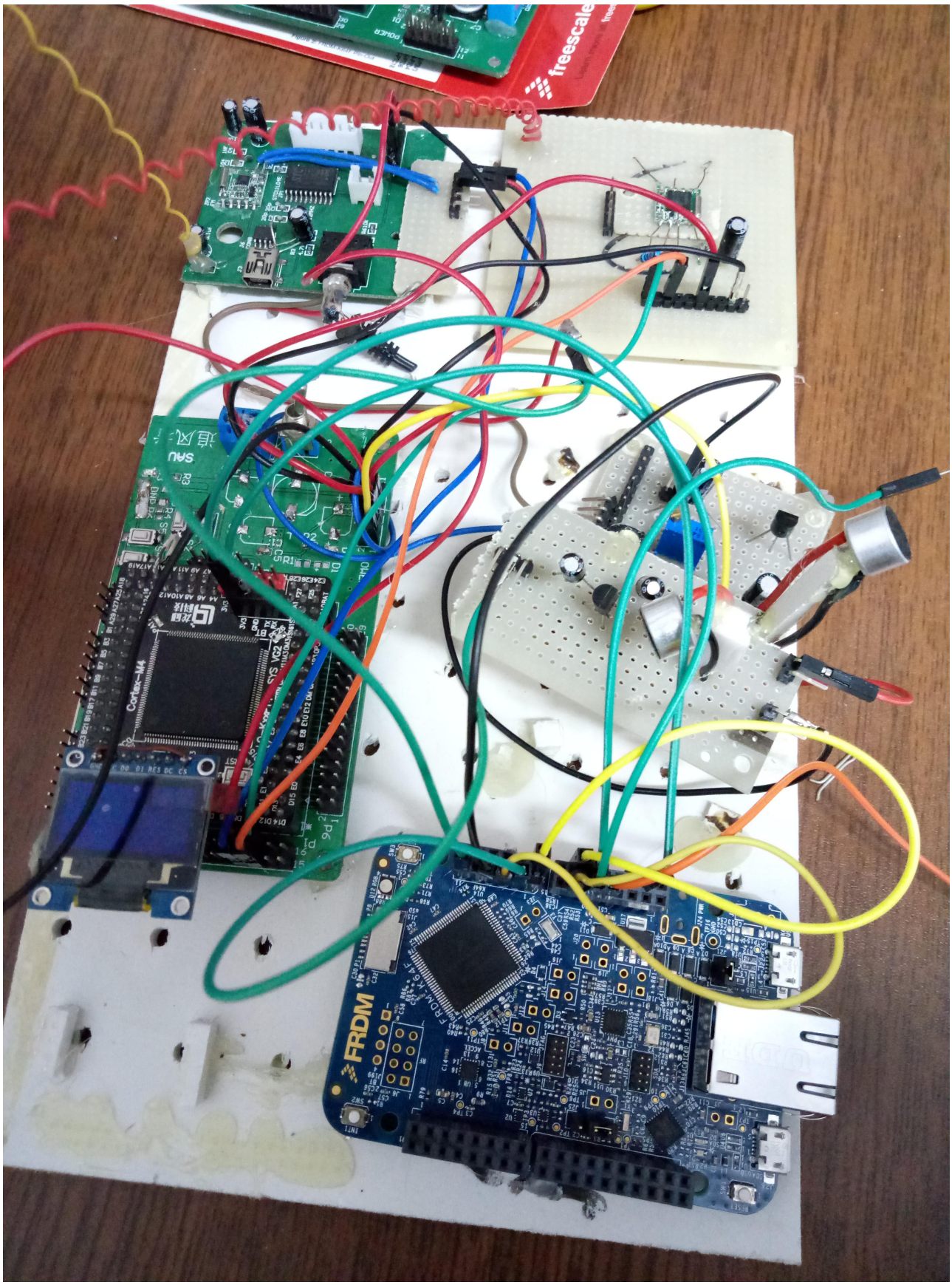
第六部分　附录

附录Ⅰ PCB 原理图








freescale
FRDM
Cortex-M4

附录Ⅱ 关键代码展示

```
#include "audio.h"
#include "LQ_12864.h"
#include "HW_PIT.h"
#include "HW_ADC.h"
#include "HW_DMA.h"
#include "HW_PDB.h"
#include "HW_DAC.h"
PDB_InitTypeDef pdb_init_struct;
ADC_InitTypeDef adc_init_struct;
DMA_InitTypeDef dma_init_struct;
PIT_InitTypeDef pit0_init_struct;
DAC_InitTypeDef dac_init_struct;
#define line    512//line_2/2
#define BUFSIZE    line/2

// HP150-FS=25K-IIR-5
// this file was generated using
// function STM32F4_iirsos_coeffs.m
#define NUM_SECTIONS 3
float b[NUM_SECTIONS][3] = {
{3.02479474E-01, -6.04895605E-01, 3.02479474E-01},
```

```
{9.80319800E-01, -1.96055729E+00, 9.80319800E-01},
{3.10203786E+00, -3.10203786E+00, 0.00000000E+00} };
float a[NUM_SECTIONS][3] = {
{1.00000000E+00, -1.99234431E+00, 9.93774392E-01},
{1.00000000E+00, -1.95906791E+00, 9.62128981E-01},
{1.00000000E+00, -8.83042076E-01, 0.00000000E+00} };
// LP3500-FS=25K-IIR-10
// this file was generated using
// function STM32F4_iirsos_coeffs.m

#define NUM_SECTIONS1 5
float b1[NUM_SECTIONS1][3] = {
{7.58725855E-01, -8.25631892E-01, 7.58725855E-01},
{7.65349173E-01, -6.85844755E-01, 7.65349173E-01},
{5.91696610E-01, -1.84846267E-01, 5.91696610E-01},
{6.23799826E-01, 8.45954815E-01, 6.23799826E-01},
{4.16672019E-03, -4.79209618E-03, 4.16672019E-03} };

float a1[NUM_SECTIONS1][3] = {
{1.00000000E+00, -1.29823469E+00, 9.51026904E-01},
{1.00000000E+00, -1.36992684E+00, 8.87623257E-01},
{1.00000000E+00, -1.49535923E+00, 7.86223316E-01},
```

```
{1.00000000E+00, -1.61884036E+00, 6.88217514E-01},
{1.00000000E+00, -1.27466020E+00, 9.86592359E-01} };
float32_t coeffs0[5*NUM_SECTIONS] =
{0},coeffs1[5*NUM_SECTIONS] = {0},coeffs2[5*NUM_SECTIONS1]
= {0},coeffs3[5*NUM_SECTIONS1] =
{0},coeffs4[5*NUM_SECTIONS1] = {0};
float32_t state0[4*NUM_SECTIONS] =
{0},state1[4*NUM_SECTIONS] = {0},state2[4*NUM_SECTIONS1] =
{0},state3[4*NUM_SECTIONS1] = {0},state4[4*NUM_SECTIONS1]
= {0};
float32_t coeffs5[5*NUM_SECTIONS] =
{0},state5[4*NUM_SECTIONS] = {0};

arm_biquad_casd_df1_inst_f32
iir_S0,iir_S1,iir_S2,iir_S3,iir_S4,iir_S5;
arm_lms_instance_f32 lmsNorm_instance;
//arm_lms_norm_instance_f32 lmsNorm_instance;

#define BETA 4e-12f // adaptive learning rate
#define NUM_TAPS 32 // number of weights
#define BLOCK_SIZE BUFSIZE
```

```
float32_t beta = 1E-11;
float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
float32_t firCoeffs32[NUM_TAPS] = {0.0};



uint8_t d2=0,Result_half=0,Result_Complete=0;//??????
uint8_t d1=0,Result_half1=0,Result_Complete1=0;
int16_t Result[line],Result1[line];
float32_t Result_deal_out[line];
uint16_t    s = BUFSIZE;//??????512????DMA????
float32_t signoise[line], yn=0, yn1=0,yn2=0,yn3=0,error[line];
float32_t input=0,input1=0,cs=0,cs1=0;

void adc_pdb_dma_init(void);
void pit0_isr(void);
void init_LED1() {    // init leds

gpio_set_mode(LEDR, Output,kPORT_MuxAsGpio);
gpio_set_mode(LEDG, Output,kPORT_MuxAsGpio);
gpio_set_mode(LEDB, Output,kPORT_MuxAsGpio);
gpio_set(LEDR, HIGH);
gpio_set(LEDG, HIGH);
```

```
gpio_set(LEDB, HIGH);
}
void dma0_half_Complete_isr(void)
{

        if(d2==0)//half_isr
        {
            Result_half =1;//Result_half deal
        }
       else//Complete_isr
        {
            Result_Complete = 1;//Result_Complete deal
            LPLD_DMA_LoadDstAddr(DMA_CH0,Result);
        }
            d2 = (d2+1)%2;
}
void dma1_half_Complete_isr(void)
{

        if(d1==0)//half_isr
        {
            //gpio_toggle(PTB20);//test
```

```
            Result_half1 =1;//Result_half deal
        }
        else//Complete_isr
        {
            Result_Complete1 = 1;//Result_Complete deal
            LPLD_DMA_LoadDstAddr(DMA_CH1,Result1);
        }
            d1 = (d1+1)%2;
}

/*
 * ???ADC????
 *
 */
void adc_pdb_dma_init(void)
{
   //??ADC????
   adc_init_struct.ADC_Adcx = ADC0;
   adc_init_struct.ADC_DiffMode = ADC_DIFF;          //????
   adc_init_struct.ADC_BitMode = DIFF_16BIT;        //??13???
   adc_init_struct.ADC_SampleTimeCfg      =      SAMTIME_SHORT;
//?????
```

```
    adc_init_struct.ADC_HwAvgSel = HW_4AVG;          //4?????
    adc_init_struct.ADC_CalEnable = true; //???????
    adc_init_struct.ADC_HwTrgCfg = HW_TRGA;         //?????????
    adc_init_struct.ADC_DmaEnable = true;           //??DMA
    //???ADC
    LPLD_ADC_Init(adc_init_struct);
    //???????????,??????ADC??
    //RUSH?????????DAD1??????
    //??DAD1??????(ADC0_DP1?ADC0_DM1)
    LPLD_ADC_Chn_Enable(ADC0, DAD1);
    //??ADC0?DAD1?????????????
    LPLD_ADC_EnableConversion(ADC0, DAD1, 0, false);
     //ADC1
     adc_init_struct.ADC_Adcx = ADC1;
     LPLD_ADC_Init(adc_init_struct);
     LPLD_ADC_Chn_Enable(ADC1, DAD1);
     LPLD_ADC_EnableConversion(ADC1, DAD1, 0, false);
    //2.??DMA
    //???PDB????
    dma_init_struct.DMA_CHx = DMA_CH0;     //??Ch0??
    dma_init_struct.DMA_Req              =              ADC0_DMAREQ;
//DMA????ADC0
```

```
    dma_init_struct.DMA_MajorLoopCnt = line;          //?????line?
    dma_init_struct.DMA_MinorByteCnt                  =                  2;
//?????????(??ADC???12?,????2??)
    dma_init_struct.DMA_SourceAddr  =  (uint32_t)&(ADC0->R[0]);
//???:ADC0?????A??
    dma_init_struct.DMA_SourceDataSize      =      DMA_SRC_16BIT;
//?????????16?
    dma_init_struct.DMA_DestAddr          =          (uint32_t)Result;
//????
    dma_init_struct.DMA_DestDataSize      =      DMA_DST_16BIT;
//??????????16?
    dma_init_struct.DMA_DestAddrOffset                =                  2;
//??????:??????1
      dma_init_struct.DMA_MajorCompleteIntEnable = true;
      dma_init_struct.DMA_MajorHalfCompleteIntEnable = true;
    dma_init_struct.DMA_AutoDisableReq                =                  false;
//????????,???????????
      dma_init_struct.DMA_Isr = dma0_half_Complete_isr;
    //???DMA
    LPLD_DMA_Init(dma_init_struct);
    //??DMA??
    LPLD_DMA_EnableReq(DMA_CH0);//????????DMA
```

```
   //????????
    //
LPLD_DMA_LoadDstAddr(DMA_CH0,Result);//??DMA??????????
   LPLD_DMA_EnableIrq(dma_init_struct);
    //dma
    dma_init_struct.DMA_CHx = DMA_CH1;
    dma_init_struct.DMA_Req = ADC1_DMAREQ;
    dma_init_struct.DMA_MajorLoopCnt = line;
//?????line?
   dma_init_struct.DMA_MinorByteCnt = 2;
//?????????(??ADC???12?,????2??)
   dma_init_struct.DMA_SourceAddr = (uint32_t)&(ADC1->R[0]);
//???:ADC0?????A??
   dma_init_struct.DMA_SourceDataSize = DMA_SRC_16BIT;
//?????????16?
   dma_init_struct.DMA_DestAddr = (uint32_t)Result1;
//????
   dma_init_struct.DMA_DestDataSize = DMA_DST_16BIT;
//???????????16?
   dma_init_struct.DMA_DestAddrOffset = 2;
//??????:??????1
    dma_init_struct.DMA_MajorCompleteIntEnable = true;
```

```
    dma_init_struct.DMA_MajorHalfCompleteIntEnable = true;
   dma_init_struct.DMA_AutoDisableReq        =        false;
//????????,???????????
    dma_init_struct.DMA_Isr = dma1_half_Complete_isr;
    LPLD_DMA_Init(dma_init_struct);
    LPLD_DMA_EnableReq(DMA_CH1);//????????DMA
    LPLD_DMA_EnableIrq(dma_init_struct);
    //dac
    dac_init_struct.DAC_Dacx = DAC0;
    LPLD_DAC_Init(dac_init_struct);
   //3.??PDB????
   //???PDB????
   pdb_init_struct.PDB_CounterPeriodMs = 1000;    //PDB???????
   pdb_init_struct.PDB_LoadModeSel = LOADMODE_0; //??????
   pdb_init_struct.PDB_ContinuousModeEnable        =        false;
//????????
   pdb_init_struct.PDB_TriggerInputSourceSel   =   TRIGGER_PIT0;
//??????PIT0
   //???PDB
   LPLD_PDB_Init(pdb_init_struct);

   //??PDB??ADC??:??ADC0?????A???????????0
```

```
    LPLD_PDB_AdcTriggerCfg(ADC0, PRETRIG_EN_A, 0);
     //adc1
     LPLD_PDB_AdcTriggerCfg(ADC1, PRETRIG_EN_A, 0);
     //4.??PIT????
    //???PIT??
    pit0_init_struct.PIT_Pitx = PIT0;        //??PIT0
    pit0_init_struct.PIT_PeriodUs = 40;   //PIT0????
   // pit0_init_struct.PIT_PeriodMs = 100;   //PIT0????1s
    pit0_init_struct.PIT_Isr = pit0_isr;   //??????
    //???PIT0
    LPLD_PIT_Init(pit0_init_struct);
    //??PIT0?PIT1
    LPLD_PIT_EnableIrq(pit0_init_struct);



void pit0_isr(void)
{
    LPLD_DAC_SetBufferDataN(DAC0,
(int16_t)(Result_deal_out[s]/32+1024 ), 1);
    s = (s+1)%line;

}
```

```
int     main(void)
{
    uint16_t i=0,k=0,s=0,y=0;
    CLOCK_SetXtal0Freq(50000000);//key clock pll init,
    init_LED1();
    gpio_set_mode(PTB20, Output,kPORT_MuxAsGpio);
   gpio_set(PTB20, HIGH);
    //LCD_Init();
    adc_pdb_dma_init();
    for (i=0; i<NUM_SECTIONS ; i++)//hp150
            { //adc0
                coeffs0[k++] = b[i][0];
                coeffs0[k++] = b[i][1];
                coeffs0[k++] = b[i][2];
                coeffs0[k++] = -a[i][1];
                coeffs0[k++] = -a[i][2];
                //adc1
                coeffs1[s++] = b[i][0];
                coeffs1[s++] = b[i][1];
                coeffs1[s++] = b[i][2];
                coeffs1[s++] = -a[i][1];
                coeffs1[s++] = -a[i][2];
```

```
    //result
    coeffs5[y++] = b[i][0];
    coeffs5[y++] = b[i][1];
    coeffs5[y++] = b[i][2];
    coeffs5[y++] = -a[i][1];
    coeffs5[y++] = -a[i][2];
}
k=0;
s=0;
y=0;
for (i=0; i<NUM_SECTIONS1 ; i++)//lp3500
{ //adc0
    coeffs2[k++] = b1[i][0];
    coeffs2[k++] = b1[i][1];
    coeffs2[k++] = b1[i][2];
    coeffs2[k++] = -a1[i][1];
    coeffs2[k++] = -a1[i][2];
    //adc1
    coeffs3[s++] = b1[i][0];
    coeffs3[s++] = b1[i][1];
    coeffs3[s++] = b1[i][2];
    coeffs3[s++] = -a1[i][1];
```

```
            coeffs3[s++] = -a1[i][2];
        //result
            coeffs4[y++] = b1[i][0];
            coeffs4[y++] = b1[i][1];
            coeffs4[y++] = b1[i][2];
            coeffs4[y++] = -a1[i][1];
            coeffs4[y++] = -a1[i][2];
        }
    /* Initialize the LMSNorm data structure */
  arm_biquad_cascade_df1_init_f32(&iir_S0,      NUM_SECTIONS,
coeffs0, state0);
    arm_biquad_cascade_df1_init_f32(&iir_S1,     NUM_SECTIONS,
coeffs1, state1);
    arm_biquad_cascade_df1_init_f32(&iir_S2,    NUM_SECTIONS1,
coeffs2, state2);
    arm_biquad_cascade_df1_init_f32(&iir_S3,    NUM_SECTIONS1,
coeffs3, state3);
    arm_biquad_cascade_df1_init_f32(&iir_S4,    NUM_SECTIONS1,
coeffs4, state4);
    arm_biquad_cascade_df1_init_f32(&iir_S5,     NUM_SECTIONS,
coeffs5, state5);
```

```
    arm_lms_init_f32(&lmsNorm_instance, NUM_TAPS, firCoeffs32,
firStateF32, beta, BUFSIZE);
   while(1)
    {
     if(Result_half ==1&&Result_half1 ==1)//deal half
        {
            gpio_set(PTB20, HIGH);
            //deal data
    for (i=0 ; i< BUFSIZE; i++)//first
    {
        input = (float32_t)(Result[i]);
     input1 = (float32_t)(Result1[i]);
        arm_biquad_cascade_df1_f32(&iir_S0, &input, &yn, 1);
        arm_biquad_cascade_df1_f32(&iir_S1, &input1, &yn1, 1);

        arm_biquad_cascade_df1_f32(&iir_S2, &yn, &yn2, 1);
        arm_biquad_cascade_df1_f32(&iir_S3, &yn1, &yn3, 1);

        signoise[i] = yn2 + input1;
        //arm_add_f32(&yn2[i],&yn3[i],&signoise[i],1);

          arm_lms_f32(&lmsNorm_instance, /* LMSNorm instance
```

```
*/
        &yn2,                                   /* Input signal */
        &signoise[i],                                   /* Reference
Signal */
        &cs,                        /* Converged Signal */
        &error[i],                              /* Error Signal, this will
become small as the signal converges */
          1);                           /* BlockSize */
      arm_biquad_cascade_df1_f32(&iir_S4,               &error[i],
&Result_deal_out[i] ,1);
      //arm_biquad_cascade_df1_f32(&iir_S5,               &cs1,
&Result_deal_out[i], 1);
   //  if(abs((int)(Result_deal_out[i]))<500)
      //  Result_deal_out[i]=0;


   }

////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
          Result_half1 = 2;//deal over
           Result_half = 2;//deal over
           gpio_set(PTB20, 0);
```

```
        }
        else if(Result_Complete == 1&&Result_Complete1 ==
1)//deal another
        {
////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////
            //deal data

            for (i= BUFSIZE ; i < line ; i++)
            {
                input = (float32_t)(Result[i]);
                input1 = (float32_t)(Result1[i]);

                arm_biquad_cascade_df1_f32(&iir_S0, &input, &yn,
1);
                arm_biquad_cascade_df1_f32(&iir_S1,       &input1,
&yn1, 1);

                arm_biquad_cascade_df1_f32(&iir_S2, &yn, &yn2,
1);
                arm_biquad_cascade_df1_f32(&iir_S3, &yn1, &yn3,
1);
```

```
            signoise[i] = yn2 + input1;
             //arm_add_f32(&yn2[i],&yn3[i],&signoise[i],1);

        arm_lms_f32(&lmsNorm_instance, /* LMSNorm instance */
          &yn2,                              /* Input signal */
          &signoise[i],                              /* Reference
Signal */
          &cs,                             /* Converged Signal */
          &error[i],                          /* Error Signal, this will
become small as the signal converges */
              1);

       arm_biquad_cascade_df1_f32(&iir_S4,          &error[i],
&Result_deal_out[i], 1);
              //      arm_biquad_cascade_df1_f32(&iir_S5,      &cs1,
&Result_deal_out[i], 1);
      // if(abs((int)(Result_deal_out[i]))<500)
         //  Result_deal_out[i]=0;
    }
```

```
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
            Result_Complete = 2;//deal over
            Result_Complete1 = 2;//deal over
        }
    };
}
```