# ANC\_control软件架构

**目录**

[ANC\_control软件架构 1](#_Toc13842285)

[1 基础说明 2](#_Toc13842286)

[2 软件框架 3](#_Toc13842287)

[3 设计框架 5](#_Toc13842288)

[3.1 芯片控制 5](#_Toc13842289)

[3.2 滤波器设计 7](#_Toc13842290)

[3.3 数字EQ设计 7](#_Toc13842291)

[4 滤波器计算详细算法 12](#_Toc13842292)

[4.1 一阶低通滤波器 12](#_Toc13842293)

[4.2 一阶高通滤波器 12](#_Toc13842294)

[4.3 带阻滤波器 13](#_Toc13842295)

[4.4 低频增强滤波器 15](#_Toc13842296)

[4.5 高频增强滤波器 17](#_Toc13842297)

[4.6 一阶低通滤波器（带运放） 18](#_Toc13842298)

[4.7 二阶低通滤波器（带运放） 20](#_Toc13842299)

[4.8 复合滤波器（带运放） 24](#_Toc13842300)

[5 界面配置 26](#_Toc13842301)

[5.1 芯片控制界面配置保存及导入 26](#_Toc13842302)

[5.2 滤波器设计界面配置保存及导入 27](#_Toc13842303)

[6 其他功能 29](#_Toc13842304)

[6.1 芯片控制（待完成） 29](#_Toc13842305)

[6.2 滤波器设计 29](#_Toc13842306)

## 基础说明

* 1. 软件基于脚本语言python v3.7.1版本编写，其他版本python可能会遇到版本兼容问题，软件目前已迁移到x86平台，支持win7/10-32/64位系统；

Python下载：<https://www.python.org/>。

* 1. 使用到的python核心库如下表，安装好python之后以下所有库均可在命令行下用pip install+库名来安装。

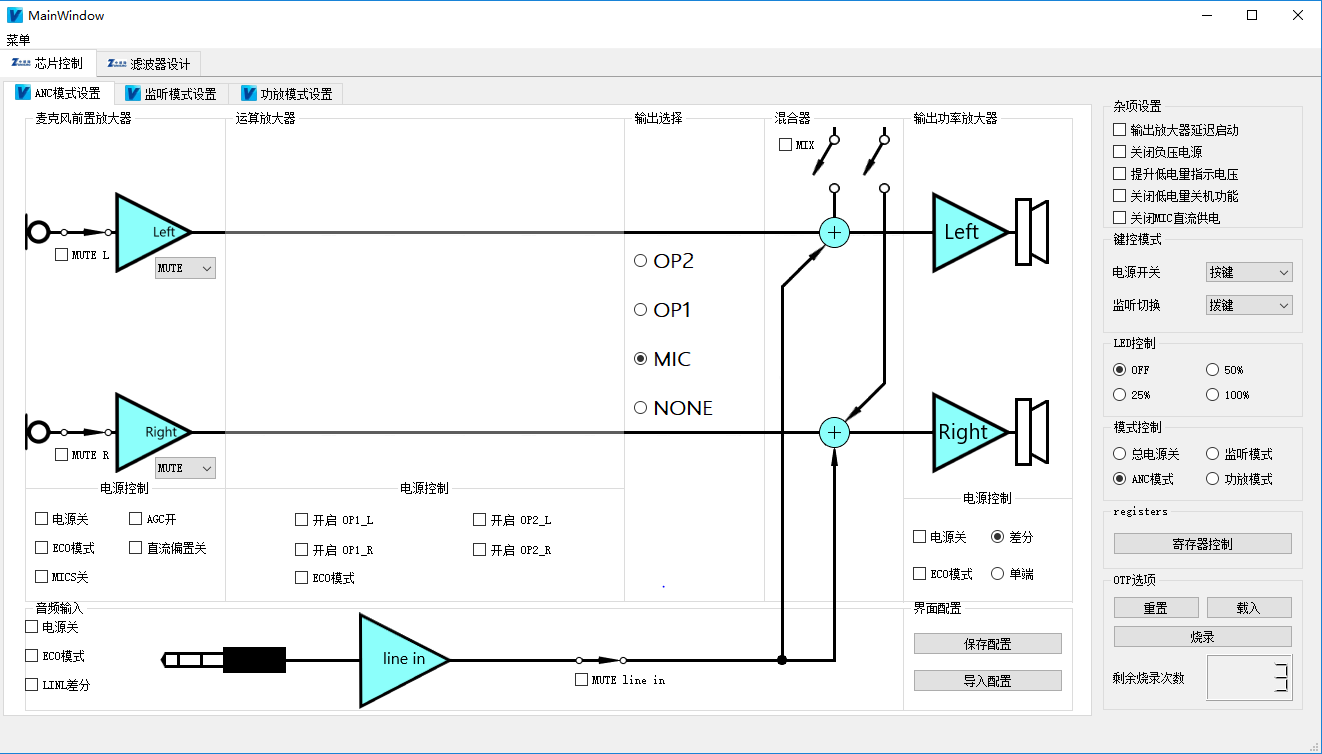
|  |  |  |
| --- | --- | --- |
| **python库名** | **版本号** | **功能简介** |
| PyQt5 | 5.11.3 | UI界面管理 |
| PySpice | 1.3.2 | 模拟电路仿真计算 |
| control | 0.8.1 | 控制系统设计 |
| numpy | 1.15.4 | 科学计算 |
| pyqt5-tools | 5.11.3.1.4 | UI设计 |
| scipy | 1.2.0 | 科学计算 |
| matplotlib | 3.0.2 | 数据绘图 |
| xlrd | 1.2.0 | excel数据读取 |
| xlwt | 1.3.0 | excel数据保存 |
| bitarray | 0.8.3 | 二进制数列控制 |

* 1. PySpice库运行需ngspice.dll（ngspice-27 32位）文件支持，安装参考<https://pyspice.fabrice-salvaire.fr/installation.html>。下载链接http://ngspice.sourceforge.net/experimental/ngspice-27\_32.7z。
  2. 软件发布之后编译为anc\_control.exe可执行文件。

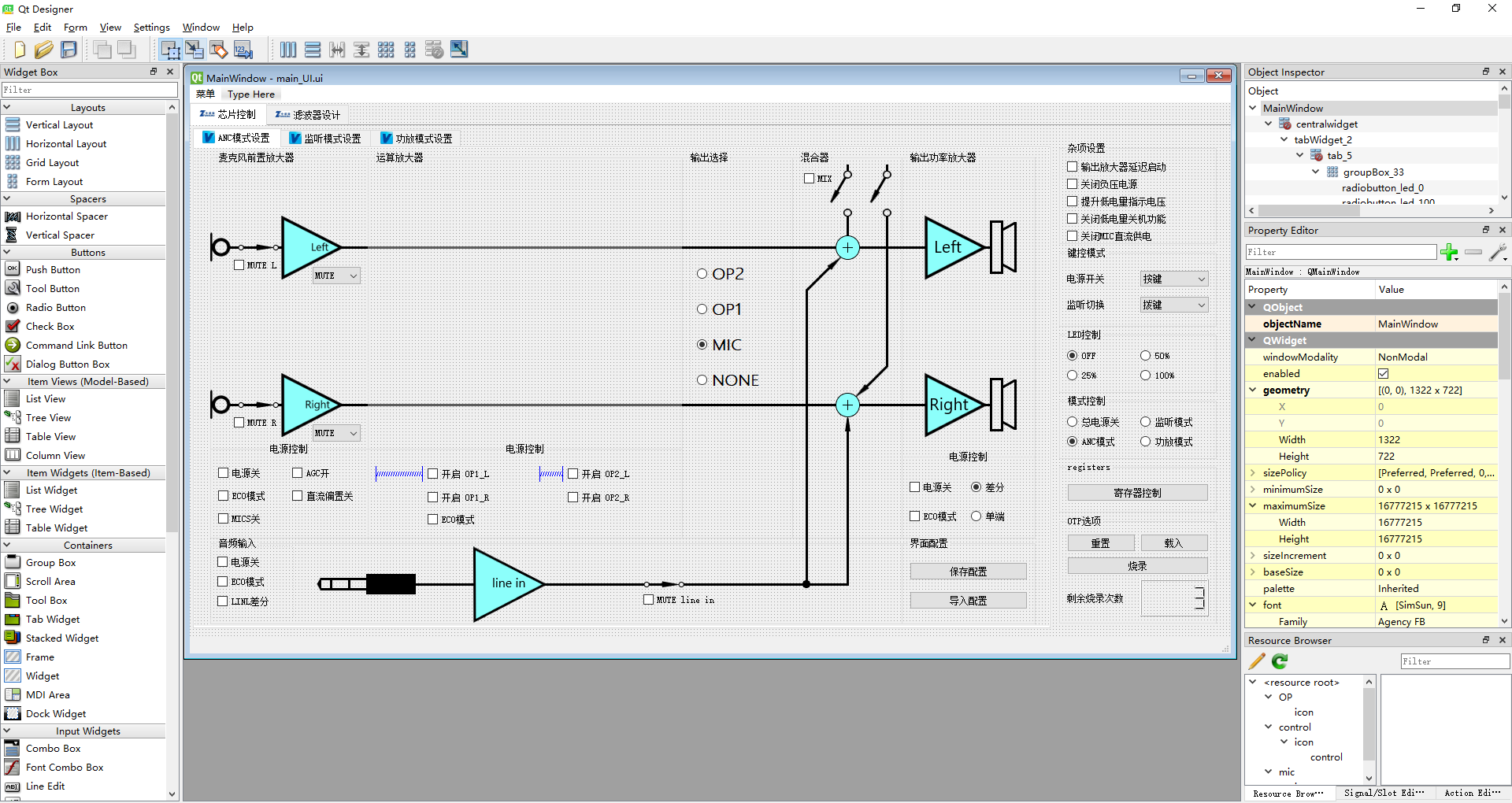
## 软件框架

软件python类继承关系框架如图：

1. 用户界面主要采用PyQt5设计框架，界面控制跟逻辑框架相互分离。QMainWindow类为框架基础父类，Ui\_MainWindow类为界面控制类，其他类FB\_Filter\_main、EQ\_Filter\_main等均为逻辑控制的类。
2. 主脚本文件anc\_control.py，集成芯片registers控制、滤波器设计等所有功能；芯片registers控制逻辑需在该文件AncControlWindow类中调整。运行后软件界面参考如下：



1. 主UI控制文件main\_UI.ui，可用pyqt5-tools下Qt-designer.exe工具打开进行用户交互界面调整。用户界面调整后用pyUIC工具编译为main\_UI.py文件，文件包含Ui\_MainWindow类为界面控制类。

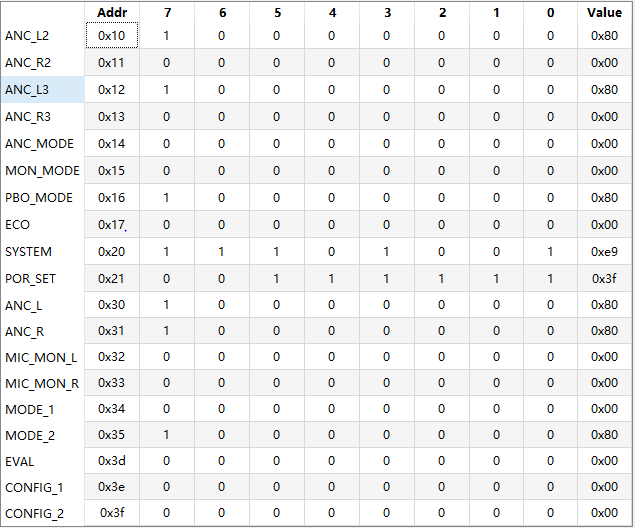


1. ANC滤波器设计主文件为filter\_designer.py，模拟滤波器设计计算核心算法集中在此文件，同时运算放大器01UI界面交互控制逻辑算法也集中在此文件。此文件也可单独运行，单独运行时只支持滤波器设计功能，芯片registers控制功能失效。
2. 芯片registers弹出对话框UI及控制逻辑集中在\register目录；其中：
   1. register\_UI.ui文件为对话框UI控制文件；
   2. register\_main.py文件为逻辑控制文件。
3. 模拟滤波电路创建、excel数据导入及仿真数据绘图函数均放在\includes目录下。其中：
   1. My\_ANC\_ExcelRead.py文件包含目标滤波器曲线导入功能；
   2. my\_filter\_simulator.py文件包含netlist模拟电路网络创建类及伯德图绘图类；
   3. my\_filter\_simulator.py文件集中编写子电路串并联拼接完成netlist电路网络函数，其中create\_netlist函数返回值为可被pyspice仿真器识别进行电路任意节点输出仿真。
   4. My\_OperationalAmplifier.py文件中定义了一个理想运算放大器，目前电路中运算放大器均采用理想运放，小于10Hz低频及高于20Hz高频仿真结果准确度偏低。
   5. my\_bode\_diagram.py文件定义了对仿真结果进行bode图绘制函数，定义相位显示区间±180°。
4. 电路参数手动调整功能集中定义在\value\_tuning文件夹，其中：
   1. notch\_tuning\_UI.ui、 notch\_tuning\_main.py两个文件控制带阻滤波器参数手动调整功能；
   2. peak\_tuning\_UI.ui、peak\_tuning\_main.py两个文件控制混合滤波器参数手动调整功能；
   3. two\_order\_lowpass\_tuning\_UI.ui、two\_order\_lowpass\_tuning\_main.py两个文件控制二阶低通滤波器参数手动调整功能。
5. 数字EQ设计功能定义在EQ\_Filter\_main.py文件，其中：
   1. MyEqFigure类为EQ绘图函数；
   2. SetEqWindow类为UI界面控制函数。

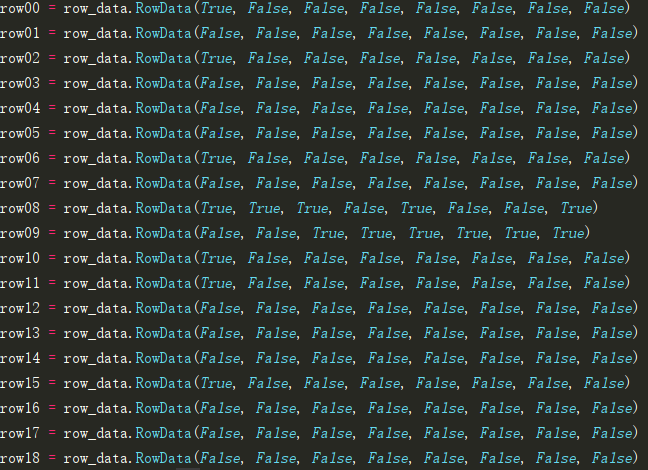
## 设计框架

### 芯片控制

* 1. 芯片已分配好的寄存器地址及功能如下图：

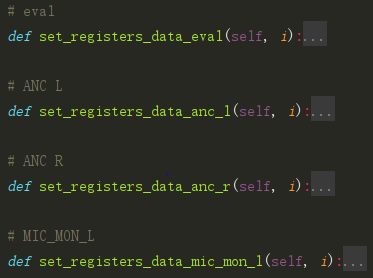


对照寄存器地址及功能表，创建python二进制数组表registers\_map：



数组表由布尔值组成，True代表寄存器默认写入为1，False代表寄存器默认写入为0；

与此对应可创建一系列逻辑控制函数，根据UI界面用户操作功能时发出的信号调整二进制数组表对应的布尔值，最后将二进制数组表写入芯片寄存器即可完成对芯片的实时控制。

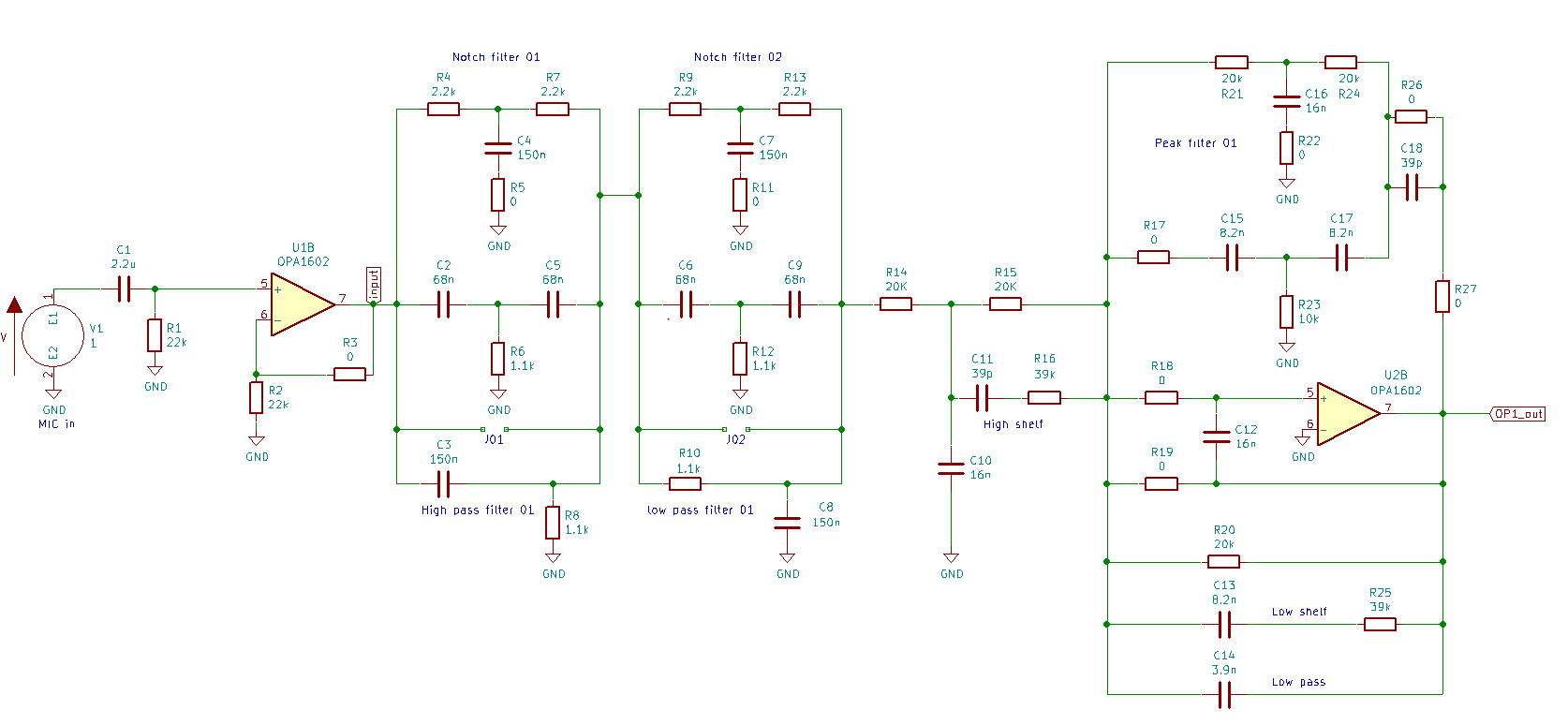


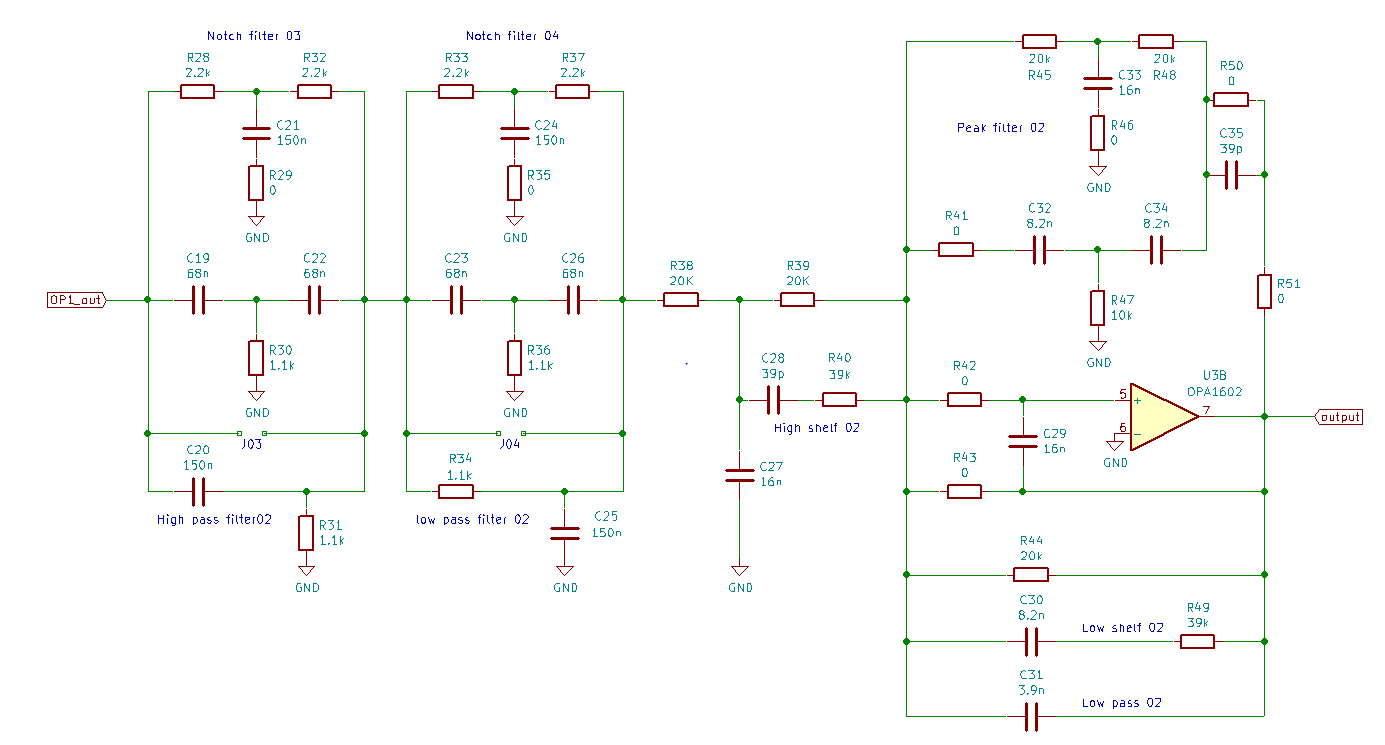
### 滤波器设计

算法基于如下电路原理图搭建：

[icon\电路原理图\_OP01.PNG](icon/电路原理图_OP01.PNG)

[icon\电路原理图\_OP02.PNG](icon/电路原理图_OP02.PNG)

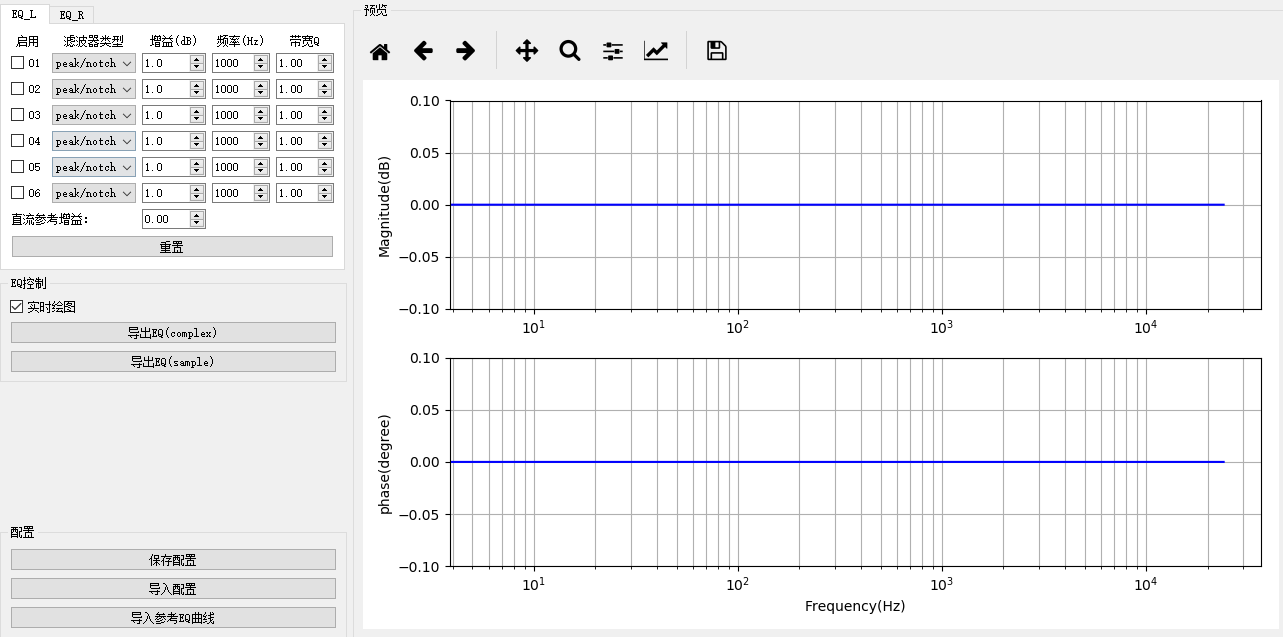




电路图中各个器件位置相对固定，电阻R及电容C最终阻容值表由软件基于UI界面用户配置生成对应BOM表，同时电路仿真也是基于生成后的BOM表进行计算仿真。

### 数字EQ设计

系统设计界面如下：



EQ设计部分算法基于以下翻译论文：

<https://blog.csdn.net/hunterhuang2013/article/details/64443718#cookbook-formulae-for-audio-eq-biquad-filter-coefficients>

基于论文内容，用python定义以下五种数字EQ滤波器：

*def* low\_pass\_filter\_iir(*f0*, *Q*=1., *fs*=192000):  
 """  
 根据PEQ参数设计二阶IIR数字低通滤波器，默认采样率192k  
 :param f0: 中心频率  
 :param Q: 峰值带宽  
 :param fs: 系统采样率  
 :return: 双二阶滤波器系数  
 """  
 w0 = 2 \* np.pi \* *f0* / *fs* alpha = np.sin(w0) / (2 \* *Q*)  
  
 b0 = (1 - np.cos(w0)) / 2  
 b1 = 1 - np.cos(w0)  
 b2 = (1 - np.cos(w0)) / 2  
 a0 = 1 + alpha  
 a1 = -2 \* np.cos(w0)  
 a2 = 1 - alpha  
 b = np.array([b0, b1, b2])  
 a = np.array([a0, a1, a2])  
 h = np.hstack((b / a[0], a / a[0]))  
  
 *return* h  
  
  
*def* high\_pass\_filter\_iir(*f0*, *Q*=1., *fs*=192000):  
 """  
 根据PEQ参数设计二阶IIR数字高通滤波器，默认采样率192k  
 :param f0: 中心频率  
 :param Q: 峰值带宽  
 :param fs: 系统采样率  
 :return: 双二阶滤波器系数  
 """  
 w0 = 2 \* np.pi \* *f0* / *fs* alpha = np.sin(w0) / (2 \* *Q*)  
  
 b0 = (1 + np.cos(w0)) / 2  
 b1 = -1 - np.cos(w0)  
 b2 = (1 + np.cos(w0)) / 2  
 a0 = 1 + alpha  
 a1 = -2 \* np.cos(w0)  
 a2 = 1 - alpha  
 b = np.array([b0, b1, b2])  
 a = np.array([a0, a1, a2])  
  
 h = np.hstack((b / a[0], a / a[0]))  
  
 *return* h  
  
  
*def* peak\_filter\_iir(*f0*, *gain*=0., *Q*=1., *fs*=192000):  
 """  
 根据PEQ参数设计二阶IIR数字peak滤波器，默认采样率192k  
 :param f0: 中心频率  
 :param gain: 峰值增益  
 :param Q: 峰值带宽  
 :param fs: 系统采样率  
 :return: 双二阶滤波器系数  
 """  
 A = np.sqrt(10 \*\* (*gain* / 20))  
 w0 = 2 \* np.pi \* *f0* / *fs* alpha = np.sin(w0) / (2 \* *Q*)  
  
 b0 = 1 + alpha \* A  
 b1 = -2 \* np.cos(w0)  
 b2 = 1 - alpha \* A  
 a0 = 1 + alpha / A  
 a1 = -2 \* np.cos(w0)  
 a2 = 1 - alpha / A  
 b = np.array([b0, b1, b2])  
 a = np.array([a0, a1, a2])  
  
 h = np.hstack((b / a[0], a / a[0]))  
  
 *return* h  
  
  
*def* low\_shelf\_filter\_iir(*f0*, *gain*=0., *Q*=1., *fs*=192000):  
 """  
 根据PEQ参数设计二阶IIR数字low shelf滤波器，默认采样率192k  
 :param f0: 中心频率  
 :param gain: 峰值增益  
 :param Q: 峰值带宽  
 :param fs: 系统采样率  
 :return: 双二阶滤波器系数  
 """  
 A = np.sqrt(10 \*\* (*gain* / 20))  
 w0 = 2 \* np.pi \* *f0* / *fs* alpha = np.sin(w0) / (2 \* *Q*)  
  
 b0 = A \* ((A + 1) - (A - 1) \* np.cos(w0) + 2 \* np.sqrt(A) \* alpha)  
 b1 = 2 \* A \* ((A - 1) - (A + 1) \* np.cos(w0))  
 b2 = A \* ((A + 1) - (A - 1) \* np.cos(w0) - 2 \* np.sqrt(A) \* alpha)  
 a0 = (A + 1) + (A - 1) \* np.cos(w0) + 2 \* np.sqrt(A) \* alpha  
 a1 = -2 \* ((A - 1) + (A + 1) \* np.cos(w0))  
 a2 = (A + 1) + (A - 1) \* np.cos(w0) - 2 \* np.sqrt(A) \* alpha  
  
 b = np.array([b0, b1, b2])  
 a = np.array([a0, a1, a2])  
  
 h = np.hstack((b / a[0], a / a[0]))  
  
 *return* h  
  
  
*def* high\_shelf\_filter\_iir(*f0*, *gain*=0., *Q*=1., *fs*=192000):  
 """  
 根据PEQ参数设计二阶IIR数字high shelf滤波器，默认采样率192k  
 :param f0: 中心频率  
 :param gain: 峰值增益  
 :param Q: 峰值带宽  
 :param fs: 系统采样率  
 :return: 双二阶滤波器系数  
 """  
 A = np.sqrt(10 \*\* (*gain* / 20))  
 w0 = 2 \* np.pi \* *f0* / *fs* alpha = np.sin(w0) / (2 \* *Q*)  
  
 b0 = A \* ((A + 1) + (A - 1) \* np.cos(w0) + 2 \* np.sqrt(A) \* alpha)  
 b1 = -2 \* A \* ((A - 1) + (A + 1) \* np.cos(w0))  
 b2 = A \* ((A + 1) + (A - 1) \* np.cos(w0) - 2 \* np.sqrt(A) \* alpha)  
 a0 = (A + 1) - (A - 1) \* np.cos(w0) + 2 \* np.sqrt(A) \* alpha  
 a1 = 2 \* ((A - 1) - (A + 1) \* np.cos(w0))  
 a2 = (A + 1) - (A - 1) \* np.cos(w0) - 2 \* np.sqrt(A) \* alpha  
  
 b = np.array([b0, b1, b2])  
 a = np.array([a0, a1, a2])  
  
 h = np.hstack((b / a[0], a / a[0]))  
  
 *return* h

用户可通过如下界面输入想要的滤波器参数：



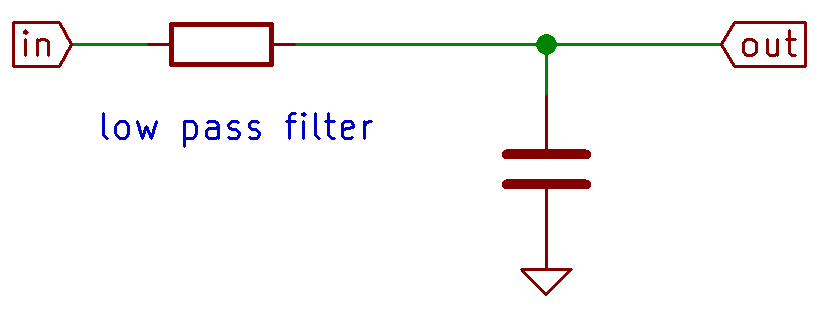
根据用户界面返回的频率f、增益gain、带宽Q、及滤波器类型四个值，调用计算函数，获得IIR滤波器的双二阶表示系数（b0,b1,b2,a0,a1,a2）;

最终滤波器参数保存结果为D12-D15定点数组。

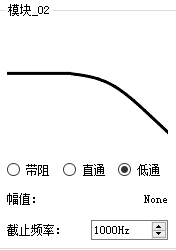
## 滤波器计算详细算法

### **一阶低通滤波器**

电路图参考如下：



UI控制界面：

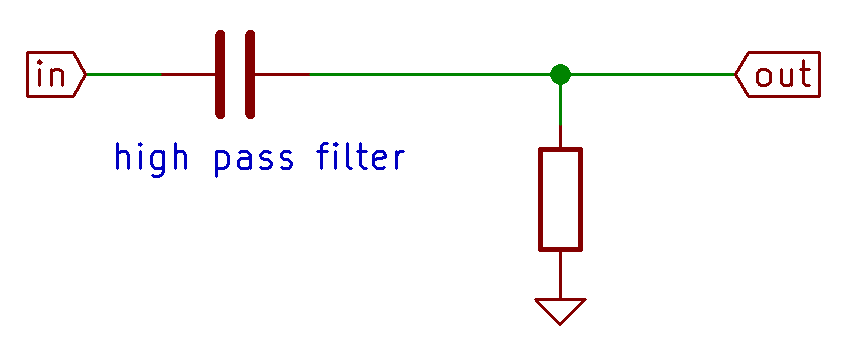
*#* *UI逻辑控制代码：*

*def* setupC\_value\_lowpass(self):  
 """根据选择的中心频率设置电容容值"""  
 f = self.frequencySpin\_lowpass.value() # 用户设置频率数值  
 c\_value = 1 / (2 \* math.pi \* 2200 \* f) # 默认电阻2.2k，c\_value为浮点进度数值  
 self.one\_order\_lowpass\_data['C\_value'] = c\_value2map(c\_value) # E12电容标准取值，str格式,ex:33nf

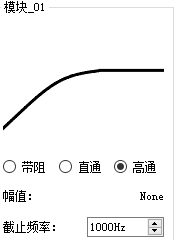
截止频率由用户选定，R值固定设置为2.2k，C值由公式计算所得，3dB低通截止频率计算公式：f=2\*pi\*R\*C。

### **一阶高通滤波器**

电路图参考如下：



UI控制界面：

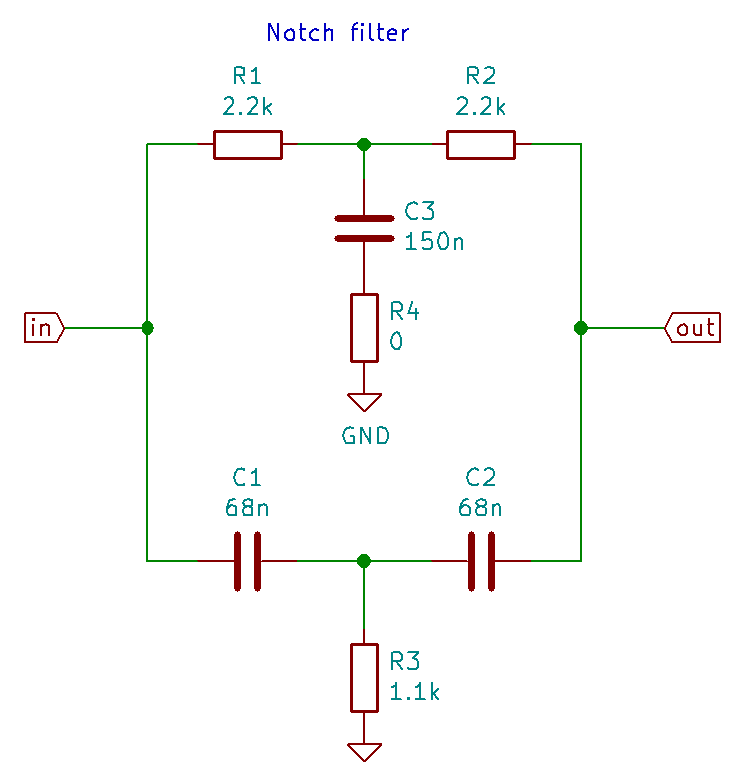
*# UI逻辑控制代码：*

*def* setupC\_value\_highpass(self):  
 """根据选择的中心频率设置电容容值"""  
 f = self.frequencySpin\_highpass.value() # 用户设置频率数值  
 c\_value = 1 / (2 \* math.pi \* 2200 \* f) # 默认电阻2.2k，c\_value为浮点进度数值  
 self.one\_order\_highpass\_data['C\_value'] = c\_value2map(c\_value) # E12电容标准取值，str格式,ex:33nf

截止频率由用户选定，R值固定设置为2.2k，C值由公式计算所得，3dB低通截止频率计算公式：f=2\*pi\*R\*C。

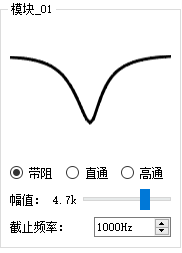
### **带阻滤波器**

电路图参考如下：



电路为双二阶RC滤波架构，为计算方便，固定3个电阻阻值R1=R2=2.2k、R3=1.1k，限制3个电容容值C1=C2=0.5\*C3；电容容值最终值由带阻中心频率确定，计算公式f=2\*pi\*R\*C；R4阻值由UI界面控制，实现滤波器幅值可调（即调整滤波器Q值）。

UI控制界面：



其中滑块幅值（图为4.7k）代表R4电阻阻值，截止频率值为带阻中心频率f值。

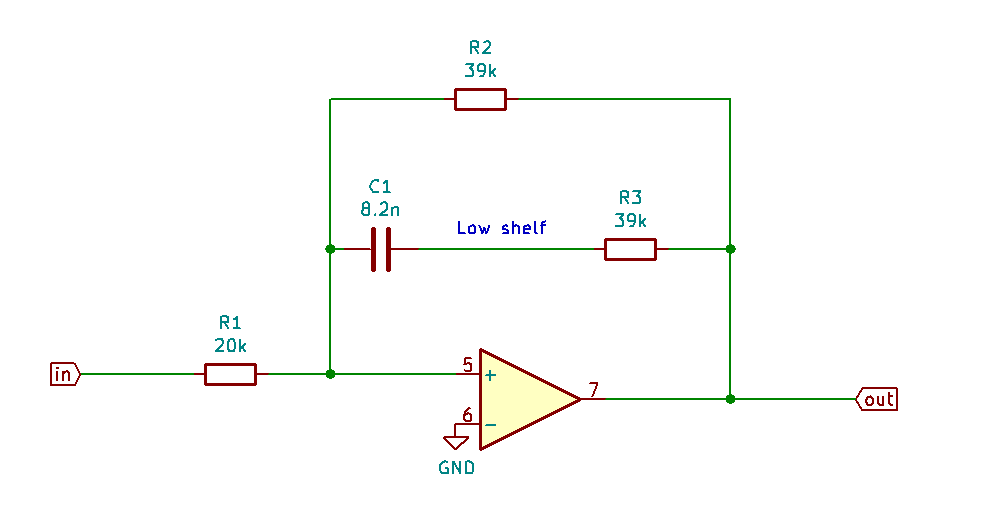
*# UI控制逻辑代码：*

*def* setupR\_value(self):  
 """  
 根据幅值滑块位置设置幅值调整电阻阻值  
 :return:  
 """  
 v = self.amplitudeSlider.value() # 滑块位置返回值  
 self.two\_order\_filter\_data['R\_gain\_value'] = get\_R\_value(v)  
  
 self.label\_Slider\_value.setText(self.two\_order\_filter\_data['R\_gain\_value']) # 设置UI界面label显示数值

*def* setupC\_value(self):  
 """  
 根据选择的中心频率设置电容容值  
 """  
 f = self.frequencySpin.value() # 用户设置频率数值  
 c\_value = 1 / (2 \* math.pi \* 2200 \* f) # 默认电阻2.2k，c\_value为浮点进度数值  
 self.two\_order\_filter\_data['C1\_value'] = c\_value2map(c\_value) # E12电容标准取值，str格式,ex:33nf  
 self.two\_order\_filter\_data['C2\_value'] = c\_value2map(c\_value)  
 self.two\_order\_filter\_data['C\_value\_double'] = c\_value2map\_dubble(c\_value) # 接地电容取并联值，str格式,ex:66nf

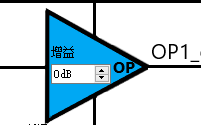
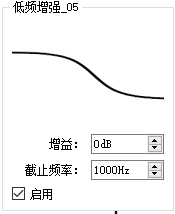
### **低频增强滤波器**

电路图参考如下：



电路为单运放反向放大电路，为计算方便，固定R1阻值为20k，固定R2=R3，电路在低频部分增益gain1=R2/R1，高频部分增益gain2=(R2|R3)/R1，3dB截止频率f=2\*pi\*R2\*C1。gain1，gain2，f三个值由UI界面控制，由此实现滤波器增益及频率可调整。

UI控制界面：



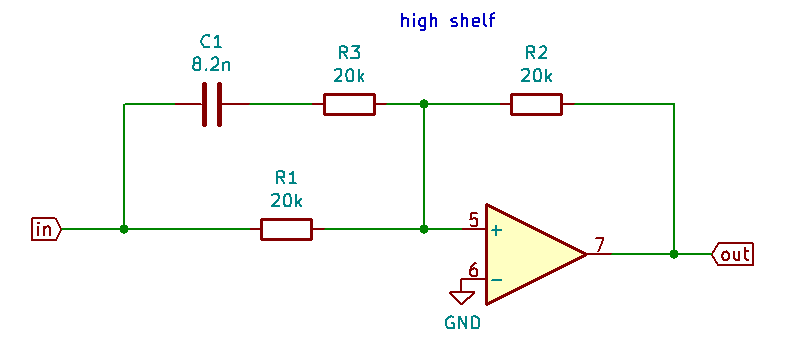
低频增益gain1为滤波器控制区输入框数字，高频增益gain2为运放控制区输入框数字，f值即为滤波器控制区截止频率数值。

*# UI控制逻辑代码：*

*def* setup\_module05\_data(self):  
 gain2 = self.low\_shelf\_gain\_spin.value()  
 gain\_op = self.OP\_gain\_spin.value()  
 *if* gain2 - gain\_op > 0:  
 Rb = 20000 \* (10 \*\* (gain2 / 20)) # Rb\_value 阻值计算  
 R2 = 20000 \* (10 \*\* ((gain2 + gain\_op) / 20) / (  
 10 \*\* (gain2 / 20) - 10 \*\* (gain\_op / 20))) # low shelf R\_value 阻值计算  
 self.op\_gain\_data['Rb\_value'] = R\_value2map(Rb)  
 self.low\_shelf\_data['R\_value'] = R\_value2map(R2)  
  
 f = self.low\_shelf\_frequency\_spin.value()  
 c\_value = 1 / (2 \* math.pi \* R2 \* f)  
 self.low\_shelf\_data['C\_value'] = c\_value2map(c\_value)

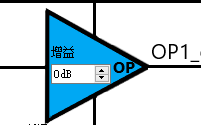
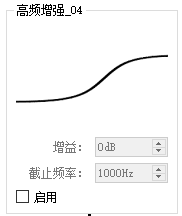
### **高频增强滤波器**

电路图参考如下：



电路为单运放反向放大电路，为计算方便，固定R1=R2=20k，电路在低频部分增益gain1=R2/R1，高频部分增益gain2=R2/(R1|R3)，3dB截止频率f=2\*pi\*R3\*C1。gain1，gain2，f三个值由UI界面控制，由此实现滤波器增益及频率可调整。

UI控制界面：



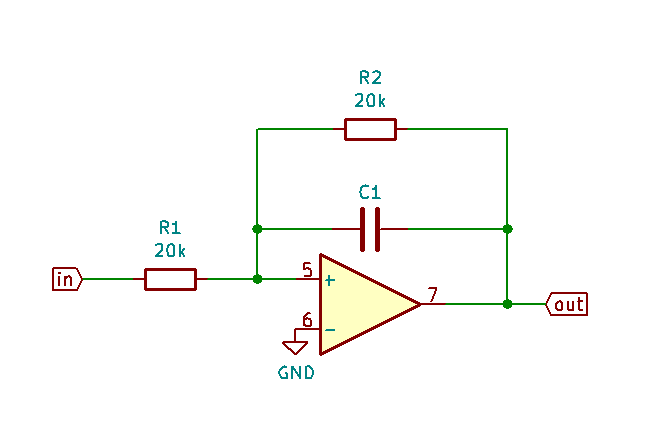
低频增益gain1为运放控制区输入框数字，高频增益gain2为滤波器控制区输入框数字，f值即为滤波器控制区截止频率数值。

*# UI控制逻辑代码：*

*def* set\_module04\_data(self):  
 gain = self.high\_shelf\_gain\_spin.value()  
 gain\_op = self.OP\_gain\_spin.value()  
 *if* gain - gain\_op > 0:  
 R1 = 20000 / (10 \*\* ((gain - gain\_op) / 20) - 1) # 反相放大器增益计算法  
 self.high\_shelf\_data['R\_value'] = R\_value2map(R1)  
  
 f = self.high\_shelf\_frequency\_spin.value()  
 c\_value = 1 / (2 \* math.pi \* R1 \* f)  
 self.high\_shelf\_data['C\_value'] = c\_value2map(c\_value)

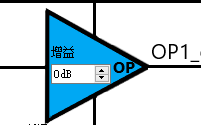
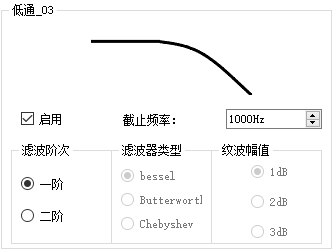
### **一阶低通滤波器（带运放）**

电路图参考如下：



电路为单运放反向放大电路，为计算方便，固定R1=20k， 系统低频部分增益gain=R2/R1, 3dB截止频率f=2\*pi\*R2\*C1，f值由UI界面控制，由此实现滤波器截止频率可调整。

UI控制界面：

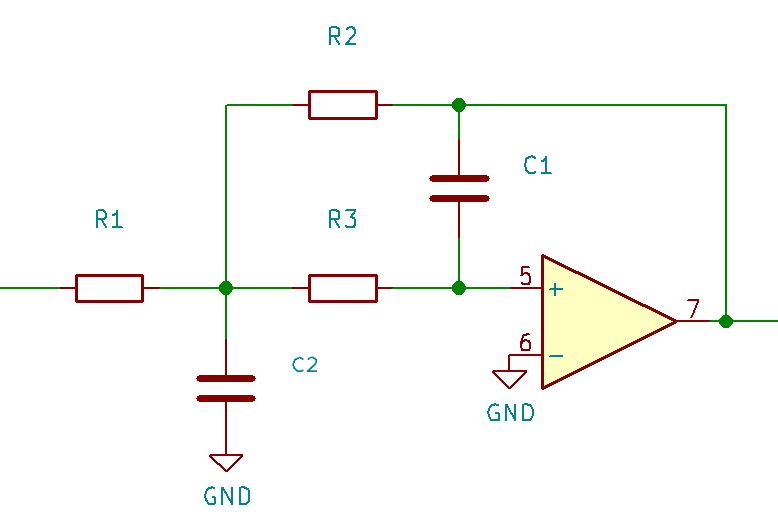


系统增益gain为运放控制区输入框数字,f值为滤波器控制区截止频率数值。

*def* set\_module\_06\_data(self):  
 gain = self.OP\_gain\_spin.value()  
 f = self.frequencySpin\_OP\_lowpass.value()  
 r\_value = get\_Rb\_value(gain)[1]  
 c\_value = 1 / (2 \* math.pi \* r\_value \* f)  
 self.op\_lowpass\_data['C\_value'] = c\_value2map(c\_value)

### **二阶低通滤波器（带运放）**

电路图参考如下：



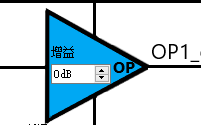
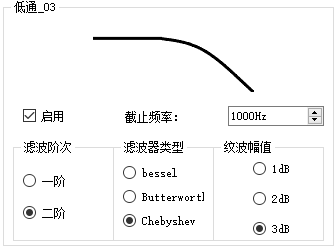
电路为单运放反向放大电路，为计算方便，固定电容C1容值为4.7nf；截止频率f由用户指定并通过UI界面控制，同时用户可选三种滤波器形式butterworth/Bessel/chebyshev，对于chebychev滤波器，用户可选择三种通道纹波幅值1/2/3dB。

以上各个状态下阻容参数计算过程：

过程参考[无限增益多路反馈有源滤波器设计](https://blog.csdn.net/openme_openwrt/article/details/7264384)，以下列出简洁过程：

1. C1电容容值由经验获得，根据系统截止频率f范围为20-20k Hz电容取值4.uf；
2. 电阻换标系数k由系统截止频率f根据公式计算所得：k=100/(f\*C1);
3. K=1时查表得出R2电阻阻值，设定为1.306k；
4. R2/R1值决定系统低频增益gain，所以R1=R2/(10\*\*(gain/20))，公式中\*\*代表次方；;
5. R3阻值为R1|R2;
6. 电容换标系数C与系统增益gain关系由该公式给出：C2=(-764.6\*gain\*\*2+1.111\*10\*\*5)/(gain\*\*2+1528);该公式根据已有测试数据拟合所得，精度有待验证；
7. Bessel模式下，电阻实际换标系数K=k，各电阻取值Ri\_value=Ri\*K，C2电容实际取值value=C\*13/68\*10\*\*(-9);
8. Batterworth模式下，电阻实际换标系数K=k\*0.9，各电阻取值Ri\_value=Ri\*K，C2电容实际取值value=C\*18/68\*10\*\*(-9);
9. Chebyshev模式下，纹波1dB状态电阻实际换标系数K=k\*0.85，各电阻取值Ri\_value=Ri\*K，C2电容实际取值value=C\*24/68\*10\*\*(-9); 纹波2dB状态电阻实际换标系数K=k\*0.8，各电阻取值Ri\_value=Ri\*K，C2电容实际取值value=C\*50/68\*10\*\*(-9); 纹波3dB状态电阻实际换标系数K=k\*075，各电阻取值Ri\_value=Ri\*K，C2电容实际取值value=C\*68/68\*10\*\*(-9);

UI控制界面：

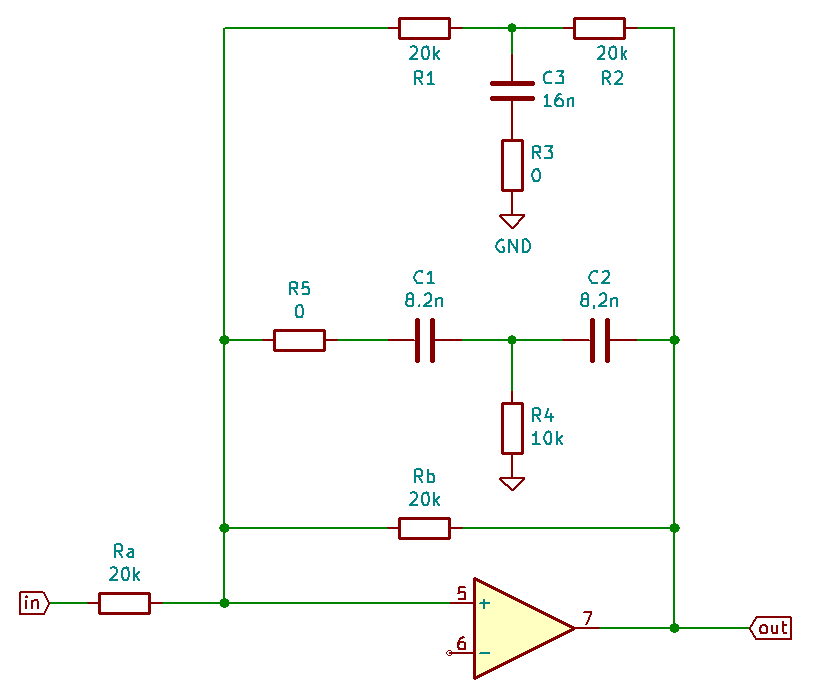


系统增益gain值为运放控制区输入框数字，截止频率f为滤波器控制区截止频率输入数值；算法代码实现如下：

*def* set\_module\_08\_data(self, *module*=*None*):  
  
 gain = self.OP\_gain\_spin.value() # 用户设置的增益  
 f = self.frequencySpin\_OP\_lowpass.value() # 用户设置的截止频率  
 C1 = 4.7e-3  
 k = 100 / (f \* C1) # 电阻换标系数  
 R2 = 1.306e3  
 R1 = R2 / (10 \*\* (gain / 20))  
 R3 = R1 \* R2 / (R1 + R2)  
 num = [-764.6, 0, 1.111e5] # 构造根据gain获得C2电容值传递函数  
 den = [1, 0, 1528]  
 Gd = control.tf(num, den)  
 x = list(range(21))  
 y = control.frd(Gd, x)  
 C = float(y.eval(gain).real)  
  
 *if module* == 'bessel':  
 K = k \* 1  
 r1 = R1 \* K  
 r2 = R2 \* K  
 r3 = R3 \* K  
 C2 = C \* 13 / 68 \* 1e-9  
 self.two\_order\_lowpass\_data['R1\_value'] = R\_value2map\_2(r1)  
 self.two\_order\_lowpass\_data['R2\_value'] = R\_value2map\_2(r2)  
 self.two\_order\_lowpass\_data['R3\_value'] = R\_value2map\_2(r3)  
 self.two\_order\_lowpass\_data['C2\_value'] = c\_value2map(C2)  
 *elif module* == 'butterworth':  
 K = k \* 0.9  
 r1 = R1 \* K  
 r2 = R2 \* K  
 r3 = R3 \* K  
 C2 = C \* 18 / 68 \* 1e-9  
 self.two\_order\_lowpass\_data['R1\_value'] = R\_value2map\_2(r1)  
 self.two\_order\_lowpass\_data['R2\_value'] = R\_value2map\_2(r2)  
 self.two\_order\_lowpass\_data['R3\_value'] = R\_value2map\_2(r3)  
 self.two\_order\_lowpass\_data['C2\_value'] = c\_value2map(C2)  
 *elif module* == 'chebyshev-1':  
 K = k \* 0.85  
 r1 = R1 \* K  
 r2 = R2 \* K  
 r3 = R3 \* K  
 C2 = C \* 24 / 68 \* 1e-9  
 self.two\_order\_lowpass\_data['R1\_value'] = R\_value2map\_2(r1)  
 self.two\_order\_lowpass\_data['R2\_value'] = R\_value2map\_2(r2)  
 self.two\_order\_lowpass\_data['R3\_value'] = R\_value2map\_2(r3)  
 self.two\_order\_lowpass\_data['C2\_value'] = c\_value2map(C2)  
 *elif module* == 'chebyshev-2':  
 K = k \* 0.8  
 r1 = R1 \* K  
 r2 = R2 \* K  
 r3 = R3 \* K  
 C2 = C \* 50 / 68 \* 1e-9  
 self.two\_order\_lowpass\_data['R1\_value'] = R\_value2map\_2(r1)  
 self.two\_order\_lowpass\_data['R2\_value'] = R\_value2map\_2(r2)  
 self.two\_order\_lowpass\_data['R3\_value'] = R\_value2map\_2(r3)  
 self.two\_order\_lowpass\_data['C2\_value'] = c\_value2map(C2)  
 *elif module* == 'chebyshev-3':  
 K = k \* 0.75  
 r1 = R1 \* K  
 r2 = R2 \* K  
 r3 = R3 \* K  
 C2 = C \* 1e-9  
 self.two\_order\_lowpass\_data['R1\_value'] = R\_value2map\_2(r1)  
 self.two\_order\_lowpass\_data['R2\_value'] = R\_value2map\_2(r2)  
 self.two\_order\_lowpass\_data['R3\_value'] = R\_value2map\_2(r3)  
 self.two\_order\_lowpass\_data['C2\_value'] = c\_value2map(C2)

### **复合滤波器（带运放）**

电路图参考如下：



电路架构较为复杂，核心框架为三个节点增益计算，滤波器峰值增益gain=Rb/Ra，低频增益gain=((R1+R2)|Rb)/Ra，高频增益gain=(R5|Rb)/Ra。

控制思路：调整R5电阻阻值可以调整滤波器高频衰减幅值，调整R3电阻阻值可以调整滤波器Q值，调整C1、C2电容容值可以控制系统中心频率。算法实现代码：

*def* set\_module\_07\_data(self):  
 self.OP\_gain\_spin.setValue(self.peak\_gain\_spin.value()) # 同步op gain & peak gain  
 gain = self.peak\_gain\_spin.value() # 滤波器峰值增益  
 f = self.peak\_frequency\_spin.value() # 中心频率  
 pos\_r\_high = self.peak\_lowpass\_slider.value() # 高频抑制电阻阻值  
 pos\_r\_gain = self.peak\_attenuation\_slider.value() # Q值控制电阻阻值  
 *if* self.peak01\_status *is False*:  
 *if* gain == 0:  
 self.module07 = *None  
 else*:  
 Rb = get\_Rb\_value(gain)[1]  
 R1 = 10000 \* Rb / (Rb - 20000)  
 c\_value = 1 / (1.414 \* math.pi \* R1 \* f) # 中心频率有1.414倍偏移  
 R3 = 2 \* R1  
 r\_high\_cut = R3 / (R3 \*\* (pos\_r\_high / 61))  
  
 self.op\_peak\_data['R1\_value'] = R\_value2map(R1)  
 self.op\_peak\_data['R2\_value'] = R\_value2map(R1)  
 self.op\_peak\_data['R\_half\_value'] = R\_value2map(R1 / 2)  
 self.op\_peak\_data['C1\_value'] = c\_value2map(c\_value)  
 self.op\_peak\_data['C2\_value'] = c\_value2map(c\_value)  
 self.op\_peak\_data['C\_value\_double'] = c\_value2map\_dubble(c\_value)  
 self.op\_peak\_data['R\_gain\_value'] = get\_R\_value(pos\_r\_gain)  
 self.op\_peak\_data['R\_high\_cut'] = R\_value2map\_2(r\_high\_cut)

## 界面配置

### **芯片控制界面配置保存及导入**

1. 保存：registers保存方式暂定为xml格式文件，文件格式为三列，nr代表在UI界面的行数，addr代表寄存器的地址，val代表寄存器应该写入的数值。

<Settings Version="1.0.0.0">  
 <Registers>  
 <reg addr="0x20" nr="8" val="0xe9" />  
 <reg addr="0x21" nr="9" val="0x3f" />  
 <reg addr="0x3d" nr="16" val="0x00" />  
 <reg addr="0x3e" nr="17" val="0x00" />  
 <reg addr="0x3f" nr="18" val="0x00" />  
 </Registers>  
 <OTPRegisters>  
 <reg addr="0x10" nr="0" val="0x80" />  
 <reg addr="0x11" nr="1" val="0x00" />  
 <reg addr="0x12" nr="2" val="0x80" />  
 <reg addr="0x13" nr="3" val="0x00" />  
 <reg addr="0x14" nr="4" val="0x00" />  
 <reg addr="0x15" nr="5" val="0x00" />  
 <reg addr="0x16" nr="6" val="0x80" />  
 <reg addr="0x17" nr="7" val="0x00" />  
 <reg addr="0x30" nr="10" val="0x80" />  
 <reg addr="0x31" nr="11" val="0x80" />  
 <reg addr="0x32" nr="12" val="0x00" />  
 <reg addr="0x33" nr="13" val="0x00" />  
 <reg addr="0x34" nr="14" val="0x00" />  
 <reg addr="0x35" nr="15" val="0x80" />  
 </OTPRegisters>  
</Settings>

1. 导入：根据设计框架3.1.a)中registers表格，解析保存的xml文件并更改二进制数组表registers\_map参数，同时调整UI界面对应的控制部件。

### **滤波器设计界面配置保存及导入**

界面保存及导入均采用Qt.Qsetting类，使用方法可参考[Qsetting介绍](https://blog.csdn.net/liuqz2009/article/details/6936834)；

保存功能部分代码参考如下：

*def* SaveSettings\_FF\_clicked(self):  
 """保存filter模块配置信息"""  
  
 filename = QtWidgets.QFileDialog.getSaveFileName(self, 'save file', '.', 'ini Files (\*.ini);;All Files (\*)')  
 *if* filename[0]:  
  
 settings = QSettings(filename[0], QSettings.IniFormat)  
 # 相位及OP模式  
 settings.setValue('phase\_status\_noninverting', self.radioButton\_noninverting\_phase.isChecked())  
 settings.setValue('phase\_status\_inverting', self.radioButton\_inverting\_phase.isChecked())  
 settings.setValue('op\_status\_two', self.radioButton\_two\_op.isChecked())  
 settings.setValue('op\_status\_one', self.radioButton\_one\_op.isChecked())  
 # mic 输入增益  
 settings.setValue('mic\_gain', self.mic\_gain\_spin.value())  
 # module01  
 settings.beginGroup('module01')  
 settings.setValue('notch\_frequency', self.frequencySpin.value())  
 settings.setValue('notch\_amplitude', self.amplitudeSlider.value())  
 settings.setValue('radioButton\_notch', self.filter01\_radioButton\_notch.isChecked())  
 settings.setValue('radioButton\_bypass', self.filter01\_radioButton\_bypass.isChecked())  
 settings.setValue('radioButton\_highpass', self.filter01\_radioButton\_highpass.isChecked())  
 settings.setValue('highpass\_frequency', self.frequencySpin\_highpass.value())  
 settings.setValue('tuning\_status', self.notch01\_status)  
 settings.setValue('tuning\_data', self.two\_order\_filter\_data)  
 settings.endGroup()

界面导入功能部分代码参考如下：



## 其他功能

### 芯片控制（待完成）

* + 1. 识别及烧写

暂无此功能，等待添加，需其他部门配合完成

* + 1. EQ实时试听

暂无此功能，等待添加，需其他部门配合完成

### 滤波器设计

#### 理想滤波器曲线导入

1.0版本支持excel/txt两种格式数据导入，数据格式分为三列：第一列为x轴（即频率轴），第二列为滤波器幅值响应（单位dB）,第三列为滤波器相位响应（单位deg）；数据从第二行开始，第一行默认为注释。格式参考如下表格：

|  |  |  |
| --- | --- | --- |
| 频率Hz | 幅值dB | 相位deg |
| 5.859375 | 4.133953 | 6.866119 |
| 11.71875 | 4.479166 | -86.5293 |
| 17.57813 | 3.192984 | -77.0986 |
| … | … | … |