



天津大学

通信原理实验报告

实验 名称	无线计算器			
组员 1	专业	电子信息工程	班级	1 班
	学号	3022234100	贡献	50%
	姓名	王烨豪	成绩	
组员 2	专业	电子信息工程	班级	1 班
	学号	3022234155	贡献	50%
	姓名	高明奇	成绩	

通信原理教研组

电气自动化与信息工程学院

天津大学

2025 年 1 月 11 日

一、实验基本信息

名称	无线计算器
仪器	两台电脑
软件	Python

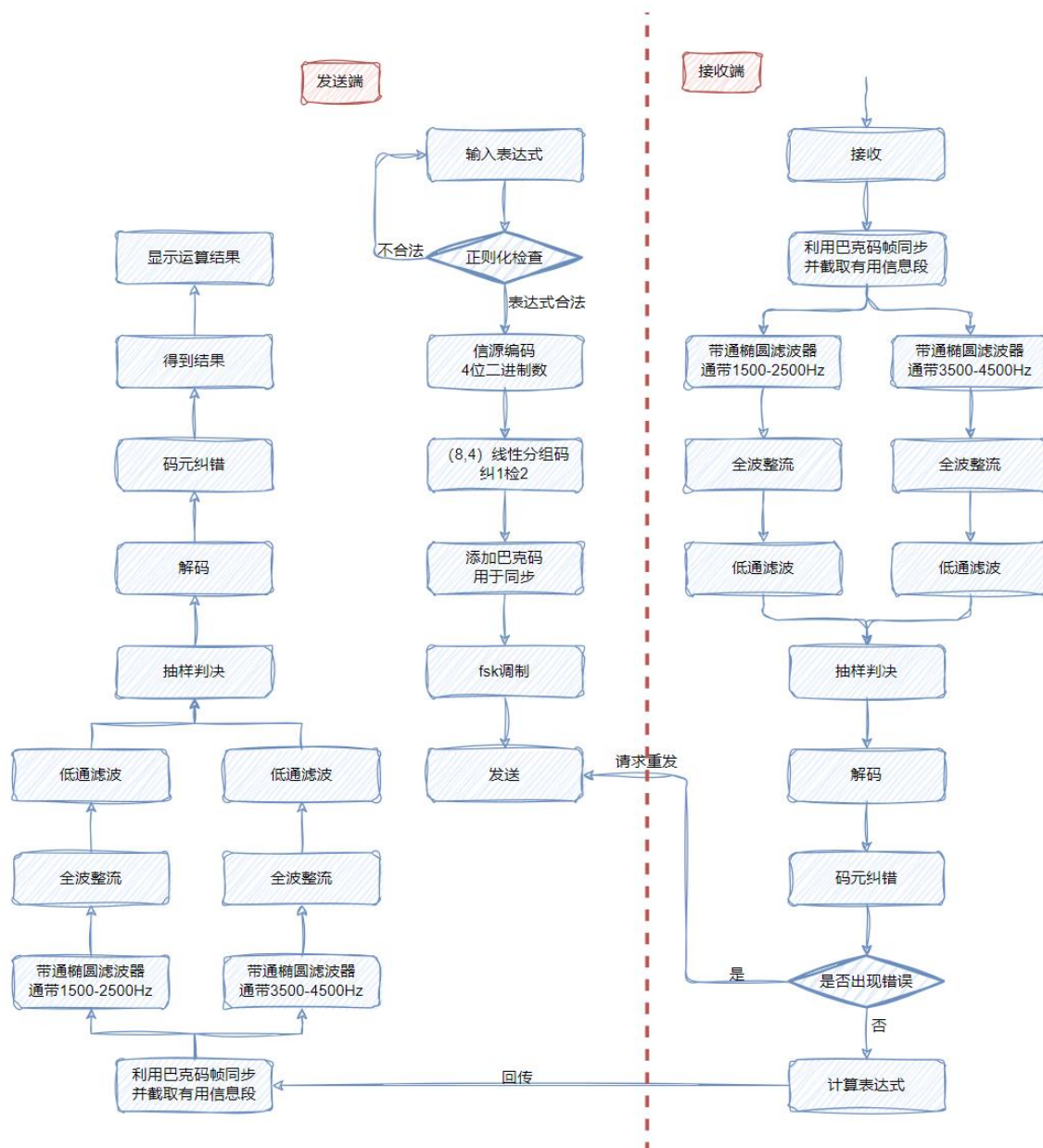
二、实验原理

1. ui 方案：采用 pyqt5，使用 qtdesigner 作为辅助设计。
2. 声波调制方案：采用 2FSK 调制，调制频率为 2000 和 4000Hz，分别表示 0 和 1。
3. 生成数字信息：在发送端的 python 中，先确认表达式合法，然后通过信源编码（每个字符用 4 位二进制数字来表示）、信道编码（(8,4) 线性分组码，纠 1 检 2），得到需要传输的数字信息。然后在最开头插入 13 位巴克码，用于获取同步信息，同时把码元数也编码发出去，便于截取有效信息。
4. 按照调制方案将其转换为声波信号，同时在头尾加上一定时间的 3000Hz 声音，作为录音开始和结束的标志。
5. 发送声波信号：将调制后的声波信号通过计算机的扬声器播放出去。
6. 接收声波信号：在接收端的 python 中，使用计算机的麦克风接收声波信号，利用巴克码获取同步信息，根据发送的有效位数截取有用信号段，以包络检波获取数字信号。
7. 解调数字信息：先尝试纠错，若可以纠正错误，则解码出表达式，通过运算显示答案。
8. 算出答案后回传，和传输表达式过程一样，只不过把头尾的 3000Hz 改成了 2500Hz 以做区分。如果运算出现错误，则回传的时候把数据位数置位 00，以请求发送端重新发送。
9. 自动录音：开启一个 40ms 的定时器，定时完成后用电脑的麦克风采取 1024 个点进行 fft，如果特定分量（3000/2500Hz）大于一定阈值，自动开始录音，当再次检测到后，再触发一个 0.2s 的一次性触发器，计时完毕后结束录音，确保有用信号录制完全。

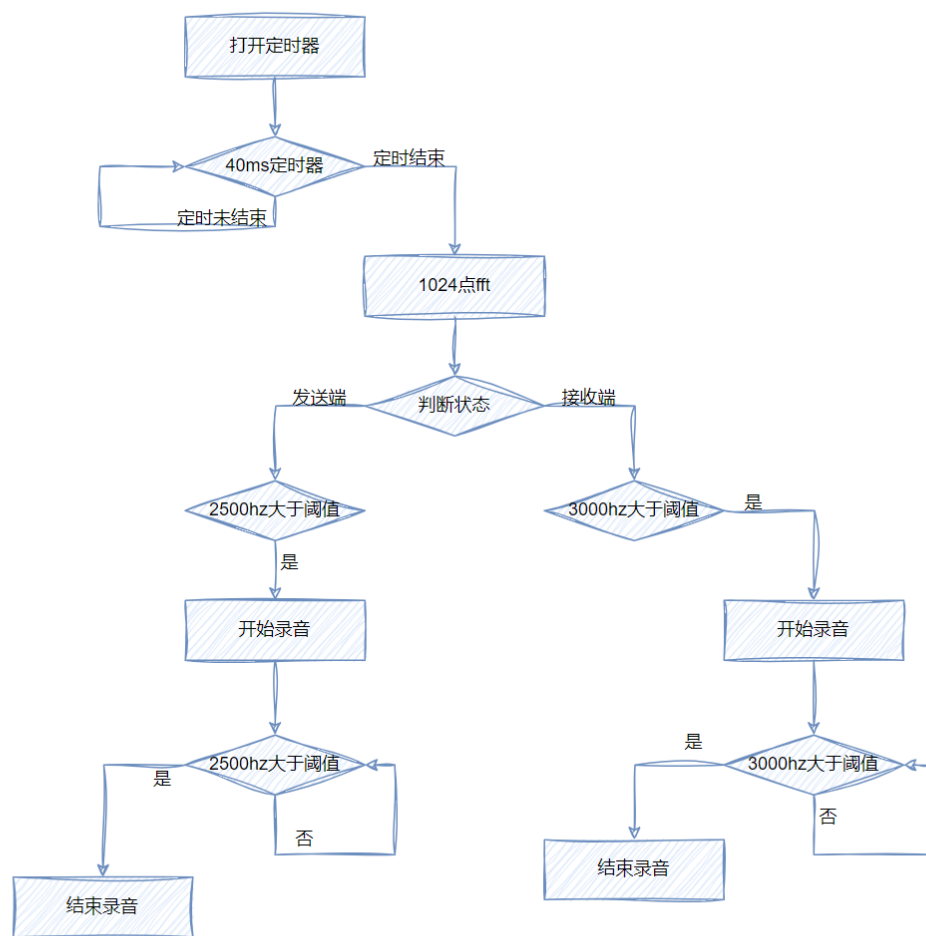
三、代码详解

3.1 总体流程图

1. 系统整体框图:

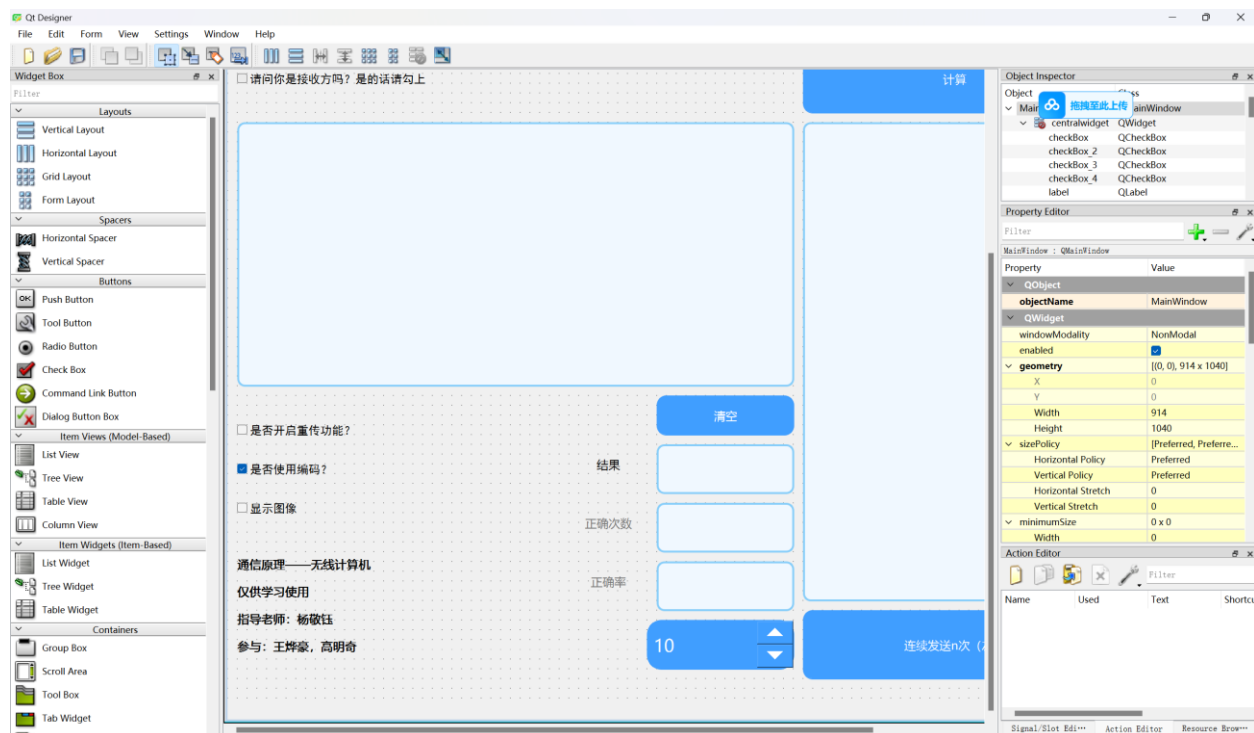


2. 录音系统框图:



3.2 ui 设计

使用 pyqt5, 在 qtdesigner 拖动并摆放好相应控件, 再转换为.py 文件。由于界面较为简单, 并没有使用布局和容器, 这意味着界面不可进行缩放。



Qss 样式代码如下：

```
QPushButton{color:#fff;background-color:#409eff;border: 1px solid #409eff;border-radius: 10px;
}QPushButton: hover{color:#fff;border-color:#66b1ff;background-color:#66b1ff;
}QPushButton:checked{color:#fff;border-color:#66b1ff;background-color: #66b1ff;
}QPushButton:pressed{color:#fff;border-color: #3a8ee6;background-color: #3a8ee6;
}
```

```
QTextEdit {
background-color: #f0f8ff; /* 背景颜色 */
color: #333; /* 字体颜色 */
border: 2px solid #87cefa; /* 边框颜色 */
border-radius: 8px; /* 圆角边框 */
padding: 10px; /* 内边距 */
font-size: 16px; /* 字体大小 */
font-family: Arial, sans-serif; /* 字体 */
}
QTextEdit:focus {
border: 2px solid #00bfff; /* 聚焦时边框颜色 */
background-color: #e6f9ff; /* 聚焦时背景颜色 */
}
```

```
QLineEdit {
background-color: #f0f8ff; /* 背景颜色 */
```

```

color: #333; /* 字体颜色 */
border: 2px solid #87cefa; /* 边框颜色 */
border-radius: 8px; /* 圆角边框 */
padding: 10px; /* 内边距 */
font-size: 16px; /* 字体大小 */
font-family: Arial, sans-serif; /* 字体 */
}

```

```

QLineEdit:focus {
border: 2px solid #00bfff; /* 聚焦时边框颜色 */
background-color: #e6f9ff; /* 聚焦时背景颜色 */
}

```

```

QSpinBox{color:#fff;background-color:#409eff;border: 5px solid #409eff;border-radius: 15px;
}

```

其他用的 12px 的微软雅黑字体。

运行界面如下：



3.3 声音检测

为了实现自动开始录音+结束录音的效果，我开了个 40ms 的定时器：

```
timer = QTimer()
timer.setInterval(40)
timer.timeout.connect(detect)
timer.start()
```

该函数每隔 40ms 运行一次，先打开麦克风读取音频流，采样率 48000hz，采 1024 个点并进行 fft。

```
def detect():
    global start_record, record_time, amplitude, count
    # if(ui.checkBox.isChecked()==True):
    p = pyaudio.PyAudio()
    stream = p.open(format=pyaudio.paInt16,
                    channels=1,
                    rate=48000,
                    input=True,
                    frames_per_buffer=1024)
    data = np.frombuffer(stream.read(1024), dtype=np.int16)
    fft_data = fft(data)
```

接收方部分，首先当音频中 3000hz 的分量超过一定阈值时，认为有用的信号开始了，于是开始录音。为了防止刚开始录音就瞬间停止，采取 0.4s 后才会触发停止录音的逻辑。在 0.4s 后，声音里 3000hz 的分量再次超过一定阈值，则停止录音并打开另外一个 0.2s 的定时器。

```
528 ~ if(ui.checkBox.isChecked()==True):#接受方
529     idx = int(3000 / freq_res)
530     amplitude = int(np.abs(fft_data[idx]) / 10240)
531     ui.textEdit_4.append("当前声音: "+str(amplitude))
532 ~ if(start_record==0):
533 ~     if(amplitude>amp):
534         record_time=time.time()
535         start_record=1
536         on_record()
537     if(start_record==1 and (time.time()-record_time)>0.4):
538 ~         if(amplitude>amp):
539 ~             start_record=2
540             timer1.start()
541
542
```

该定时器在 0.2s 后停止录音并解算结果，这是为了确保将所有的信号音频录制完成，如果结束录音太快，可能会导致有一些音节遗漏，进而造成误码错码。此外还清掉了标志位。

```

def on_timeout():
    global start_record, re_start_record, twice
    twice += 1
    if (twice == 2):
        on_stop()
        de_hammingcode()
        re_start_record = 0
        start_record = 0
        twice = 0
        timer1.stop()

```

发送方同理，只不过把所有的 3000Hz 改成了 2500Hz。

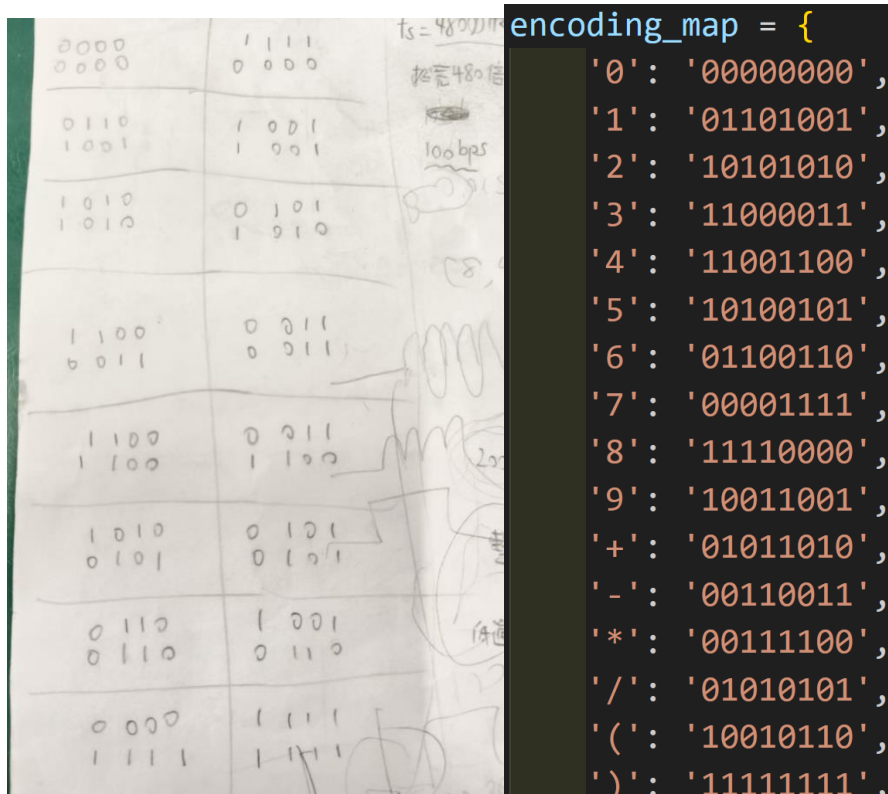
```

543     else: #发送者
544         idx = int(2500 / freq_res)
545         amplitude = int(np.abs(fft_data[idx]) / 10240)
546         ui.textEdit_4.append("当前声音: " + str(amplitude))
547         if (re_start_record == 0):
548             if (amplitude > amp):
549                 record_time = time.time()
550                 re_start_record = 1
551                 on_record()
552                 if (re_start_record == 1 and (time.time() - record_time) > 0.4):
553                     if (amplitude > amp):
554                         re_start_record = 2
555                         timer1.start()
556
557

```

3.4 编码

把 0-9 和 +-* /() 这十六个符号直接映射到最后的 (8,4) 线性分组码。



这其实是 74 汉明码并在开头加了一个奇偶校验位，这个码有个非常好的地方，解码非常非常简单，从地位向高位编号，把 1 的位置的编号全部做异或，是 0 就没有出错，不是 0 且奇偶校验位出错，如果得到的数字是 a，说明是除去奇偶校验位后，从左往右数，第 a 个数字的地方出错了，取反即可。如果不是 0 且奇偶校验位正确，则说明出错，需要重传。

比如传输的是 00001111, 含有 1 的位置是 1,2,3,4, 则 0001 异或 0010 异或 0011 异或 0100, 等于 0, 说明没有传输错误。

Python 一句话的事情。

```
index=reduce(lambda x,y:x^y,[i for i,bit in enumerate(ch) if bit])
```

下面用一个例程测试。如图，当编码正确，输出 0.

```
1 from functools import reduce
2 ch=[0,1,1,0,1,0,0,1]
3 print(reduce(lambda x,y:x^y,[i for i,bit in enumerate(ch) if bit]))
```

(tongxin) C:\Users\19144\Desktop\通信原理>

(tongxin) C:\Users\19144\Desktop\通信原理> c: && cd c:\Users\19144\Desktop\software_history\lab_py && cmd /C "e:\anaconda\install\envs\tongxin\python.exe c:\Users\19144\vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher 50272 -- C:\Users\19144\Desktop\通信原理\test.py"

(tongxin) c:\Users\19144\Desktop\software_history\lab_py>

4

如图，该码除去奇偶校验位，从左往右的第四个位置出错，程序输出 4，直接修改即可。

```
1 from functools import reduce
2 ch=[0,1,1,0,0,0,0,1]
3 print(reduce(lambda x,y:x^y,[i for i,bit in enumerate(ch) if bit]))
```

(tongxin) c:\Users\19144\Desktop\software_history\lab_py>

(tongxin) c:\Users\19144\Desktop\software_history\lab_py> c: && cd c:\Users\19144\Desktop\software_history\lab_py && cmd /C "e:\anaconda\install\envs\tongxin\python.exe c:\Users\19144\vscode\extensions\ms-python.debugpy-2024.14.0-win32-x64\bundled\libs\debugpy\adapter\..\..\debugpy\launcher 50314 -- C:\Users\19144\Desktop\通信原理\test.py"

(tongxin) c:\Users\19144\Desktop\software_history\lab_py>

4

字典里把相反的序列也放一下，便于解码。由于纠错后奇偶校验位被去掉了，因此只有七位了。

```

'0000000': '0',
'1101001': '1',
'0101010': '2',
'1000011': '3',
'1001100': '4',
'0100101': '5',
'1100110': '6',
'0001111': '7',
'1110000': '8',
'0011001': '9',
'1011010': '+',
'0110011': '-',
'0111100': '*',
'1010101': '/',
'0010110': '(',
'1111111': ')'

```

映射字典确定后，下面就是正式的编码了。我们使用的是 13 位巴克码 `barker = "1111100110101"`，在码头加入巴克码用于同步，之后发送 2 位代表字符串长度，再把字符串映射到相应的二进制数字即可。

```

hammingcode+=barker
number=len(calculation)
if(number>=10):
    for charn in str(number):
        hammingcode=hammingcode+encoder.encode(charn)
    else:
        hammingcode=hammingcode+encoder.encode('0')
        hammingcode=hammingcode+encoder.encode(str(number))
for char in calculation:
    hammingcode=hammingcode+encoder.encode(char)
ui.textEdit_2.append("已编码，码为: "+hammingcode)

```

此外，由于添加了非编码情况下的发送和接收功能，因此如法炮制也有个无编码的映射器。无编码情况下的函数总体与有编码一样，只不过映射变了。

```
# 定义编码映射
encoding_map2 = {
    '0': '0000',
    '1': '0001',
    '2': '0010',
    '3': '0011',
    '4': '0100',
    '5': '0101',
    '6': '0110',
    '7': '0111',
    '8': '1000',
    '9': '1001',
    '+': '1010',
    '-': '1011',
    '*': '1100',
    '/': '1101',
    '(': '1110',
    '0000': '0',
    '0001': '1',
    '0010': '2',
    '0011': '3',
    '0100': '4',
    '0101': '5',
    '0110': '6',
    '0111': '7',
    '1000': '8',
    '1001': '9',
    '1010': '+',
    '1011': '-',
    '1100': '*',
    '1101': '/',
    '1110': '(',
    '1111': ')'
}
```

回传信号的编码代码也类似，只不过加了如果计算结果错误，就把代表总体位数的那两位置为 00，来让发送端重新发送信号。

```
def re_encode():
    global cal
    global recode
    global barker
    recode = ''
    recode += barker
    if(ui.checkBox_3.isChecked() == True):
        if cal == 'error':
            recode = recode + encoder.encode('0')
            recode = recode + encoder.encode('0')
        else:
            number = len(str(eval(cal)))
            if(number >= 10):
                for charn in str(number):
                    recode = recode + encoder.encode(charn)
            else:
                recode = recode + encoder.encode('0')
```

3.5 生成 2fsk

采样频率 48000hz，每个二进制码元持续 0.01s。先在开头插入 0.2s 的 3000hz'音频，以激活接收方录音机开始录音。

0 发送 2000hz，1 发送 4000hz，完成 2fsk 调制，再在帧尾加上 3000hz 音频，使得接收方可以自动停止录音并采取后续的解算。

回传信号同理，只不过头尾激活音频变成了 2500Hz，函数为 re_fsk()

```
def fsk(code):
    global audio_da
    duration = 0.01
    sampling_freq = 48000 # 采样频率
    tone_freq1 = 2000
    tone_freq2 = 4000
    t = np.linspace(0, duration, int(sampling_freq * duration), endpoint=False)
    # audio_da = 32000 * np.sin(2 * np.pi * tone_freq_extra * t_extra)
    audio_data_list = [audio_da]
    # audio_data_list = []
    for co in code:
        if(co == '0'):
            audio_data_list.append(7000 * np.sin(2 * np.pi * tone_freq1 * t))
            # audio_data = np.concatenate(audio_data, 0.5 * np.sin(2 * np.pi * tone_freq1 * t))
        else:
            audio_data_list.append(32000 * np.sin(2 * np.pi * tone_freq2 * t))
            # audio_data = np.concatenate(audio_data, 0.5 * np.sin(2 * np.pi * tone_freq2 * t))

def re_fsk(code):
    t_extra = np.linspace(0, 0.2, int(48000 * 0.2), endpoint=False)
    audio_d = 32000 * np.sin(2 * np.pi * 2500 * t_extra)
    duration = 0.01
    sampling_freq = 48000 # 采样频率
    tone_freq1 = 2000
    tone_freq2 = 4000
    t = np.linspace(0, duration, int(sampling_freq * duration), endpoint=False)
    audio_data_list = [audio_d]
    # audio_data_list = []
    for co in code:
        if(co == '0'):
            audio_data_list.append(7000 * np.sin(2 * np.pi * tone_freq1 * t))
            # audio_data = np.concatenate(audio_data, 0.5 * np.sin(2 * np.pi * tone_freq1 * t))
        else:
            audio_data_list.append(32000 * np.sin(2 * np.pi * tone_freq2 * t))
            # audio_data = np.concatenate(audio_data, 0.5 * np.sin(2 * np.pi * tone_freq2 * t))
```

3.6 接收方同步与解码

接收方录音完毕后，自动进入解码函数。首先进行的是同步，生成一段和发送端发送的一模一样的巴克码音频（即 13 位巴克码按每个码元 0.01s 进行 2fsk 调制），然后与录音音频做互相关运算。由于巴克码良好的自相关特性，会在接收音频的巴克码部分产生一个相当大的互相关峰值，实测这个是非常非常准的，误差基本小于 10 个点，找到起始位置后，之后每 480 个点（采样率 48000hz，每个音频 0.01s，自然每个码元对应 480 个离散采样点）抽样判决一次，这样的抽样判决成功率非常高，几乎不会因为同步问题发生错误。没有使用教材上的采用自相关大于多少位认为是帧头，因为那样写感觉代码太麻烦了。这样直接找自相关峰值比较方便。

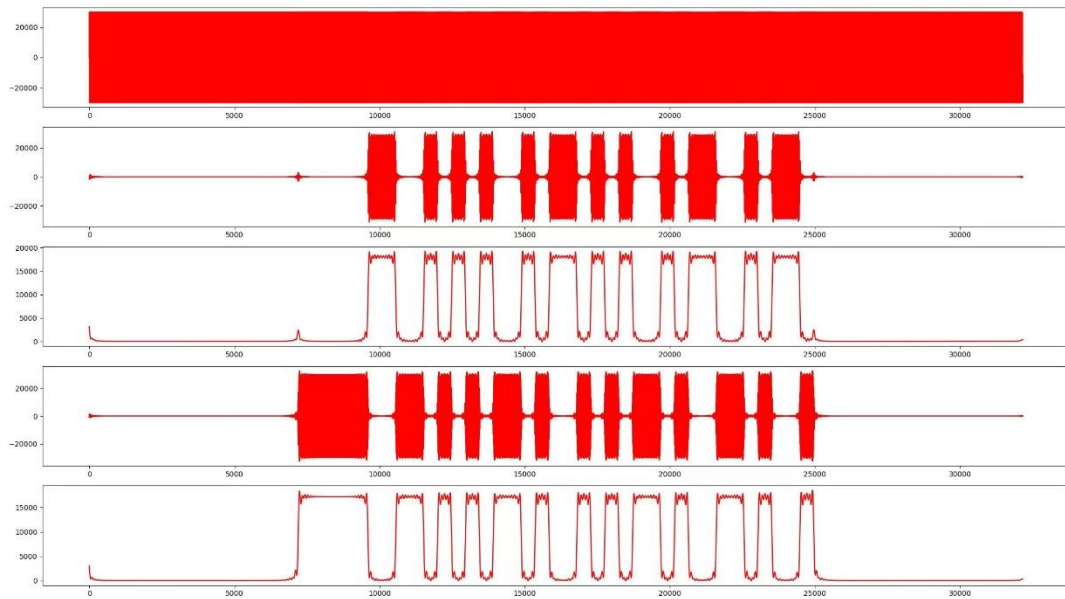
```
duration = 0.01
sampling_freq = 48000 # 采样频率
tone_freq1 = 2000
tone_freq2 = 4000
t = np.linspace(0, duration, int(sampling_freq * duration), endpoint=False)
for co in barker:
    if(co=='0'):
        barker_code.append(1 * np.sin(2 * np.pi * tone_freq1 * t))
    else:
        barker_code.append(1 * np.sin(2 * np.pi * tone_freq2 * t))
barker_code_combined = np.concatenate(barker_code)
audio_data1, framerate = read_wave('output.wav')
correlation = correlate(audio_data1, barker_code_combined, mode='same')
max_index = np.argmax(np.abs(correlation))+3120+240
ui.textEdit_2.append(f"信号的起始位置是"+str(max_index))
```

滤波器设计，通带分别为 1500-2500hz、3500-4500hz 的 5 阶带通椭圆滤波器，以及截止频率为 480hz 的低通滤波器。

```
# 带通椭圆滤波器设计，通带为[1500, 2500]
[b11,a11] = signal.ellip(5, 0.5, 60, [1500 * 2 / sampling_freq, 2500 * 2 / sam
# 低通滤波器设计，通带截止频率为480Hz
[b12,a12] = signal.ellip(5, 0.5, 60, (480 * 2 / sampling_freq), btype = 'lowpa

# 带通椭圆滤波器设计2，通带为[3500, 4500]
[b21,a21] = signal.ellip(5, 0.5, 60, [3500 * 2 / sampling_freq, 4500 * 2 / sam
```

分别通过带通滤波器，然后使用包络检波（全波整流+低通滤波）即可。下面是理想状态下的波形，可以看到，效果非常非常理想。从上到下分别是：生成的波形，通过中心 2000hz 的带通滤波器的波形，对此低通滤波后的结果，通过中心 4000hz 的带通滤波器的波形，对此低通滤波后的结果。



包络检波部分代码：

```
# 通过包络检波器
lowpass_out1 = signal.filtfilt(b12, a12, np.abs(bandpass_out1))

lowpass_out2 = signal.filtfilt(b12, a12, np.abs(bandpass_out2))
```

绘图代码

```
fig2 = plt.figure()
fig2.add_subplot(5, 1, 1)
plt.plot(np.arange(0, (len(audio_data1)),1), audio_data1, 'r')
plt.show()
fig2.add_subplot(5, 1, 2)
plt.plot(np.arange(0, (len(bandpass_out1)),1), bandpass_out1, 'r')
plt.show()
fig2.add_subplot(5, 1, 3)
plt.plot(np.arange(0, (len(lowpass_out1)),1), lowpass_out1, 'r')
plt.show()
fig2.add_subplot(5, 1, 4)
plt.plot(np.arange(0, (len(bandpass_out2)),1), bandpass_out2, 'r')
plt.show()
fig2.add_subplot(5, 1, 5)
plt.plot(np.arange(0, (len(lowpass_out2)),1), lowpass_out2, 'r')
plt.show()
```

之后每个 480 点抽样判决一次，就是 2000hz 的最终波形减去 4000hz 的最终波形，大于 0 判为 0，小于 0 判为 1。先把巴克码后紧跟的 2 个码元判出来，这个是字符串长度，然后后面再判这么多次就好了。判完后，用字典对应关系，得到字符串，如果没有错误就计算结果。


```

cal=''
if(ui.checkBox_3.isChecked()==True):
    shi=int(find_char(binary_data[0:8]))
    ge=int(find_char(binary_data[8:16]))
    num=shi*10+ge
    if(num==0 and ui.checkBox_2.isChecked()==True):
        os.system('generated_audio.wav')
        ui.textEdit_2.append("发生错误，重新传输")
    else:
        ui.textEdit_2.append("得到的码为"+str(binary_data[16:16+num*8]))
        for i in range(16, 16+num*8, 8):
            cal+=find_char(binary_data[i:i+8])
        ui.textEdit_2.append("识别到的表达式: "+cal)
        ui.textEdit_2.append("结果为"+str(eval(cal)))
        ui.lineEdit.setText(str(eval(cal)))

```

如果是没有使用编码的情况下：

```

shi=int(find_char(binary_data[0:4]))
ge=int(find_char(binary_data[4:8]))
num=shi*10+ge
if(num==0 and ui.checkBox_2.isChecked()==True):
    os.system('generated_audio.wav')
    ui.textEdit_2.append("发生错误，重新传输")
else:
    ui.textEdit_2.append("得到的码为"+str(binary_data[8:8+num*4]))
    for i in range(8, 8+num*4, 4):
        cal+=find_char(binary_data[i:i+4])
    ui.textEdit_2.append("识别到的表达式: "+cal)
    ui.textEdit_2.append("结果为"+str(eval(cal)))
    ui.lineEdit.setText(str(eval(cal)))

```

3.7 遇到错误自动重传


若遇到错误，且开启自动重传的选项框被勾上了，接收方的回传信号中信号长度变成 0，此时发送端接收到信号后就会重新发送一遍信号。


```
if(num==0 and ui.checkBox_2.isChecked()==True):
    os.system('generated_audio.wav')
    ui.textEdit_2.append("发生错误，重新传输")
```

3.8 程序打包

为了方便其他人使用，我使用了 PyInstaller 工具将其转化为 exe 文件，直接传给别人就可以直接使用，什么包都不用 import。

```
(tongxin) c:\Users\19144\Desktop\software_history\lab_py>cd C:\Users\19144\Desktop\通信原理
(tongxin) C:\Users\19144\Desktop\通信原理>pyinstaller --onefile gui.py
354 INFO: PyInstaller: 6.11.1, contrib hooks: 2824.10
354 INFO: Python: 3.8.20 (conda)
369 INFO: Platform: Windows-10-10.0.22631-SP0
369 INFO: Python environment: E:\anaconda\install\envs\tongxin
370 INFO: wrote C:\Users\19144\Desktop\通信原理\gui.spec
384 INFO: Module search paths (PYTHONPATH):
['E:\\anaconda\\install\\envs\\tongxin\\Scripts\\pyinstaller.exe',
'E:\\anaconda\\install\\envs\\tongxin\\python38.zip',
'E:\\anaconda\\install\\envs\\tongxin\\DLLs',
'E:\\anaconda\\install\\envs\\tongxin\\lib',
'E:\\anaconda\\install\\envs\\tongxin',
'E:\\anaconda\\install\\envs\\tongxin\\lib\\site-packages',
'E:\\anaconda\\install\\envs\\tongxin\\lib\\site-packages\\setuptools\\_vendor',
'C:\\Users\\19144\\Desktop\\通信原理']
824 INFO: checking Analysis
824 INFO: Building Analysis because Analysis-00.toc is non existent
825 INFO: Running Analysis Analysis-00.toc
825 INFO: Target bytecode optimization level: 0
825 INFO: Initializing module dependency graph...
827 INFO: Initializing module graph hook caches...
844 INFO: Analyzing base_library.zip ...
1572 INFO: Processing standard module hook 'hook-heapq.py' from 'E:\\anaconda\\install\\envs\\tongxin\\lib\\site-packages\\PyInstaller\\hooks'
1778 INFO: Processing standard module hook 'hook-encodings.py' from 'E:\\anaconda\\install\\envs\\tongxin\\lib\\site-packages\\PyInstaller\\hooks'
2641 INFO: Processing standard module hook 'hook-pickle.py' from 'E:\\anaconda\\install\\envs\\tongxin\\lib\\site-packages\\PyInstaller\\hooks'
3266 INFO: Caching module dependency graph...
```

 gui.exe	2024/12/16 13:43	应用程序	95,268 KB
---	------------------	------	-----------

3.9 随机表达式生成

通过 random 库，随机生成个位数表达式。

```
def generate_expression():
    # 生成个位数
    num1 = random.randint(0, 9)
    num2 = random.randint(0, 9)

    # 随机选择加号或减号
    operator = random.choice(['+', '-'])

    # 生成加减法字符串
    expression = f"{num1}{operator}{num2}"

    return expression
```

点击连续发送按钮后，标志位被置 1

```
def continue_send():  
    global continue_flag  
    continue_flag=1  
    continue_send_handle()
```

此后在到达规定次数时，不断进行互传的功能，最后通过统计的正确次数，得到成功率。

```
568 def continue_send_handle():  
569     global calculation,have_code_flag,continue_flag,continue_count,right_cc  
570     if(ui.checkBox.isChecked()==False):  
571         n=ui.spinBox.value()  
572         if(continue_count<n):  
573             calculation=generate_expression()  
574             have_code_flag=1  
575             hamming_encode()  
576             send()  
577  
578             continue_count+=1  
579         else:  
580             continue_flag=0  
581             continue_count=0  
582  
583             ui.lineEdit_2.setText(str(right_count))  
584             ui.lineEdit_3.setText(str(int(right_count/n*100))+'%')  
585             right_count=0
```

四、系统演示

发送端：

MainWindow

计算式输入

编码

发送

开始录音

停止录音

清空输入

完成输入

解码

计算

☐ 请问你是接收方吗？是的话请勾上

当前声音：1
当前声音：0
当前声音：1
当前声音：3
当前声音：2
当前声音：2
当前声音：1
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：1

结果为0
已编码，码为：
111110011010100000001100001
1011001100011001101100110
已发送
开始录音
结束录音
信号的起始位置是14691
得到的码为[0, 0, 0, 0, 0, 0, 0]
识别到的表达式：0
结果为0
已编码，码为：
111110011010100000001100001
101101001100110011111000011
已发送
开始录音
结束录音
信号的起始位置是13739
得到的码为[0, 0, 1, 1, 0, 0, 1, 1, 1, 0,
1, 0, 1, 0, 1, 0]
识别到的表达式：-2
结果为-2
开始录音

☐ 是否开启重传功能？
☒ 是否使用编码？
☐ 显示图像

通信原理——无线计算机
仅供学习使用
指导老师：杨敬钰
参与：王烨豪，高明奇

清空

结果

-2

正确次数

10

正确率

100%

10

连续发送n次（左）

MainWindow

计算式输入

编码

发送

开始录音

停止录音

清空输入

完成输入

解码

计算

☒ 请问你是接收方吗？是的话请勾上

当前声音：1
当前声音：1
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0
当前声音：0

回传结果
开始录音
结束录音
信号的起始位置是13366
得到的码为[0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0]
识别到的表达式：6-6
结果为0
回传信息已编码，码为：
111110011010100000000110100
100000000
回传结果
开始录音
结束录音
信号的起始位置是13104
得到的码为[0, 1, 1, 0, 1, 0, 0, 1, 0, 0,
1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1]
识别到的表达式：1-3
结果为-2
回传信息已编码，码为：
1111100110101000000001010101
00011001110101010
回传结果

☐ 是否开启重传功能？
☒ 是否使用编码？
☐ 显示图像

通信原理——无线计算机
仅供学习使用
指导老师：杨敬钰
参与：王烨豪，高明奇

清空

结果

-2

正确次数

正确率

10

连续发送n次（左）

20

五、成员贡献及其说明

王烨豪负责发送端代码的书写和 gui 框架的搭建。

高明奇负责 ui 的美化，接收端的代码书写。

两人共同参与了调试、试验、改进的全过程，因此贡献度各为百分之 50。