# 第二次课程设计报告

**无52 肖善誉 2015011009**

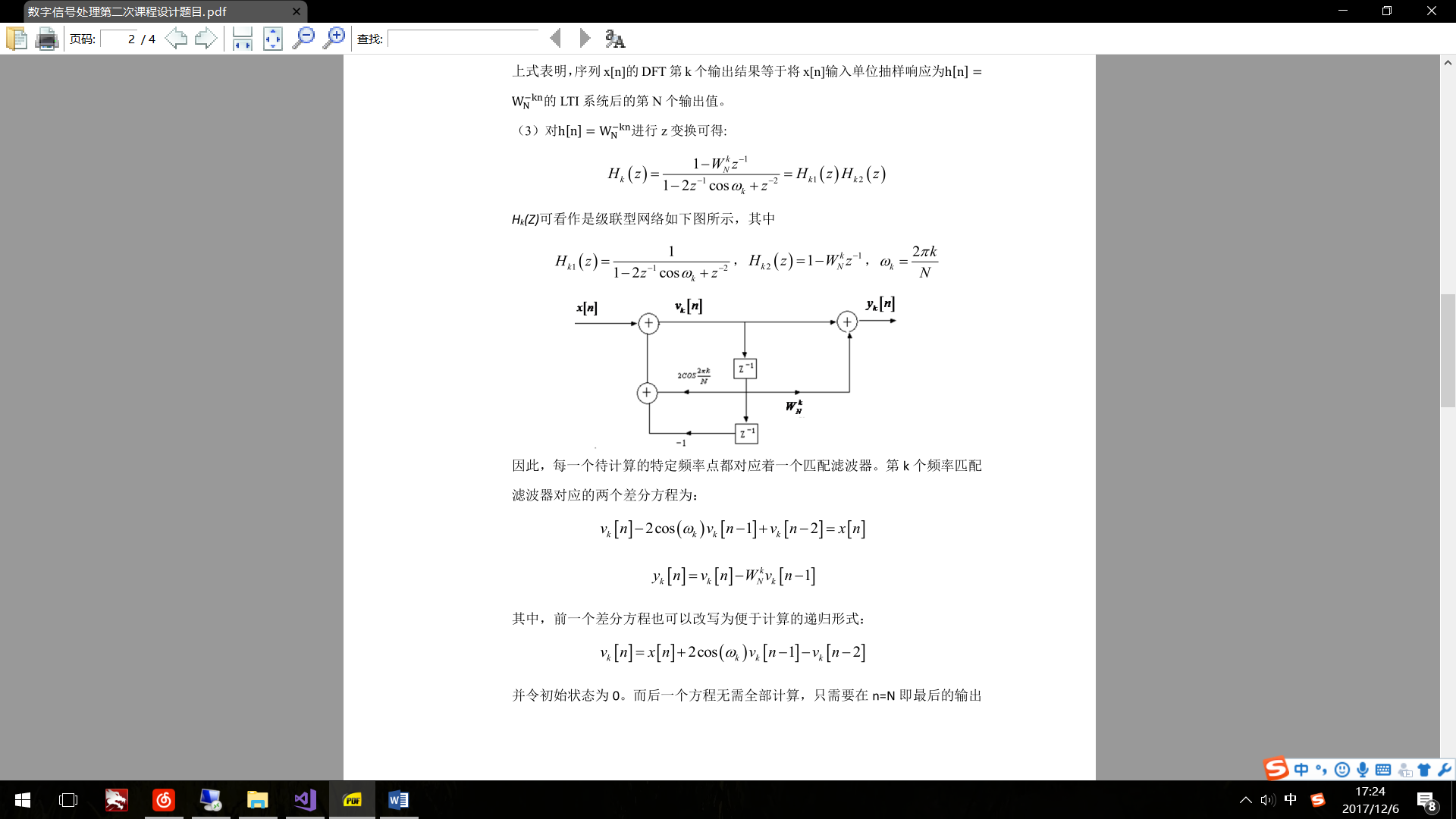
1. **程序设计思路分析**

本次实验中，我们主要通过编写程序，来对特定频率的双音多频DTMF信号进行检测。运用了如下两种算法实现：

**1.FFT**：利用FFT算法对整个时域信号做离散傅里叶变换，得到整个频谱的信息。在已知8个不同的双音多频编码频率的情况下，我们可以提取这些特定频点的幅值（或功率密度大小），以此得到对应的按键。

**2.Goertzel** ：利用相位因子的周期性，可将 DFT 定义表示成如下形式：

因此，我们可以设计一冲击响应为的滤波器，信号通过该滤波器后在时刻的值就是特定频点的离散傅里叶系数。将该滤波器转化为差分方程形式：



故该滤波器计算对应的c++函数如下：

//v: the hidden states of filter, length 2. v[0], v[1] represent vk[n - 1], vk[n - 2]

//every time the func is called, v is updated

//a1, b1: cos(omega\_k), Wk

void Goertzel\_filter(double &x, double \*v, complex<double> &y, double &a1, complex<double> &b1) {

double vn = 0.0;

vn = x + 2.0 \* a1 \* v[0] - v[1];

y = vn - b1 \* v[0];

//update state

v[1] = v[0];

v[0] = vn;

return;

}

可以看出，在该函数实现中，每个采样点n处，该滤波器进行**2+1=3次实数加法**，**2+2=4次实数乘法**（计算的差分方程只涉及实数运算，实数+复数为1次实数加法，实数\*复数为2次实数乘法）。

整个Goertzel算法函数如下：

void Goertzel(int N, int k, double \*x, complex<double> &Xk) {

//compute filter coeff

double a1 = cos(2.0 \* PI \* k / N);

complex<double> b1(cos(-2.0 \* PI \* k / N), sin(-2.0 \* PI \* k / N));

double v[2] = { 0.0, 0.0 };

complex<double> y;

for (int i = 0; i < N; i++) {

Goertzel\_filter(x[i], v, y, a1, b1);

}

double xN = 0.0;

Goertzel\_filter(xN, v, y, a1, b1);

Xk = y;

}

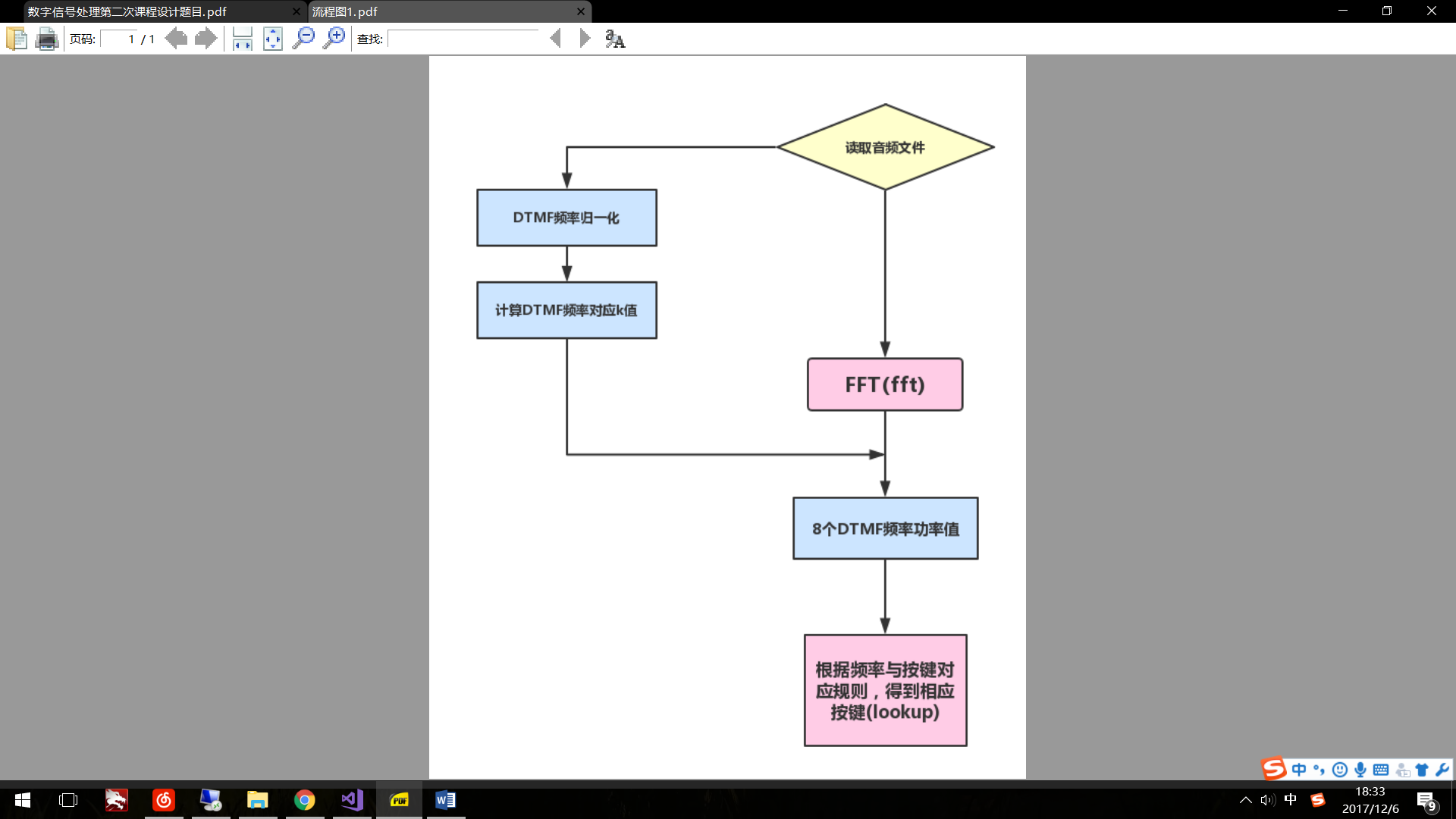
计算一个频点的频率值需要次滤波器操作。

在我们已知8个双音多频频率的前提下，我们只需要计算特定8个值处的，然后可以知道整个信号由哪两个单频信号组成，以此得到对应的按键。

1. **程序流程图**

**（1）**下载附件包中第一小题的 10 个长度不一的音频文件，利用第一次课程设计中编写的FFT程序对这 10 个文件中的DTMF信号进行频谱分析，最后给出10文件所对应的真实数字。

主要流程如下（括号中为对应的c++函数名）：



先将8个频率697.0, 770.0, 852.0, 941.0, 1209.0, 1336.0, 1477.0, 1633.0Hz归一化到。然后计算出这8个频率点在中对应的。

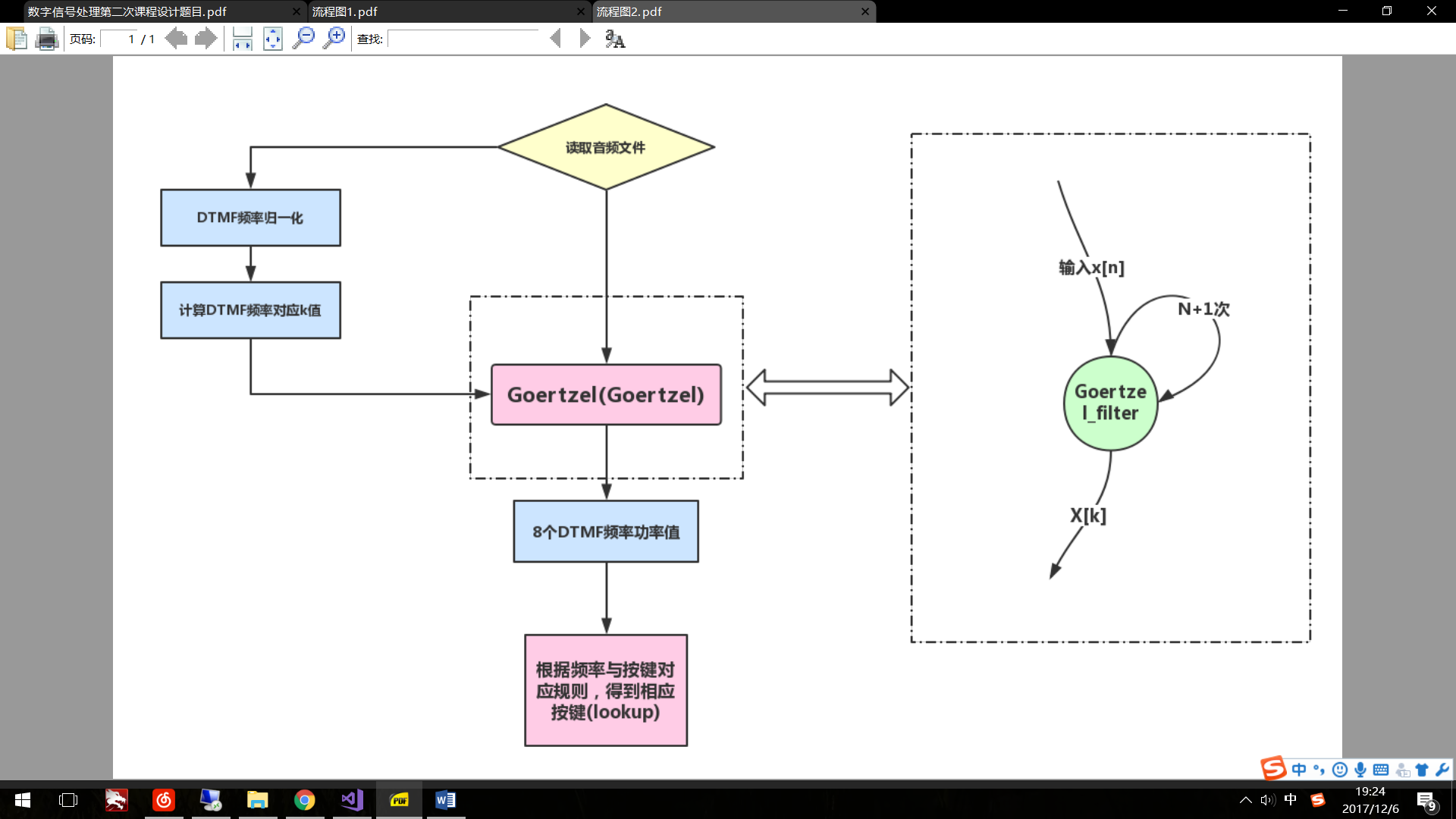
由于每次计算所得的一般不是正整数（频率离散采样点很难完全在指定的频率上采样），因此每次索引到该值上下两个整数值处的频率值，并比较它们的幅值大小，将幅值更大的那个值作为实际幅度值。

计算得到8个频点的幅度值后，分别找到行频率和列频率中幅度值最大的那个。根据频率查表(lookup函数)得对应的按键。

具体请见FFT\_anaysis函数。

**（2）**编写Goertzel算法的 C/C++语言程序，完成（1）中的要求。

主要流程如下（括号中为对应的c++函数名）：



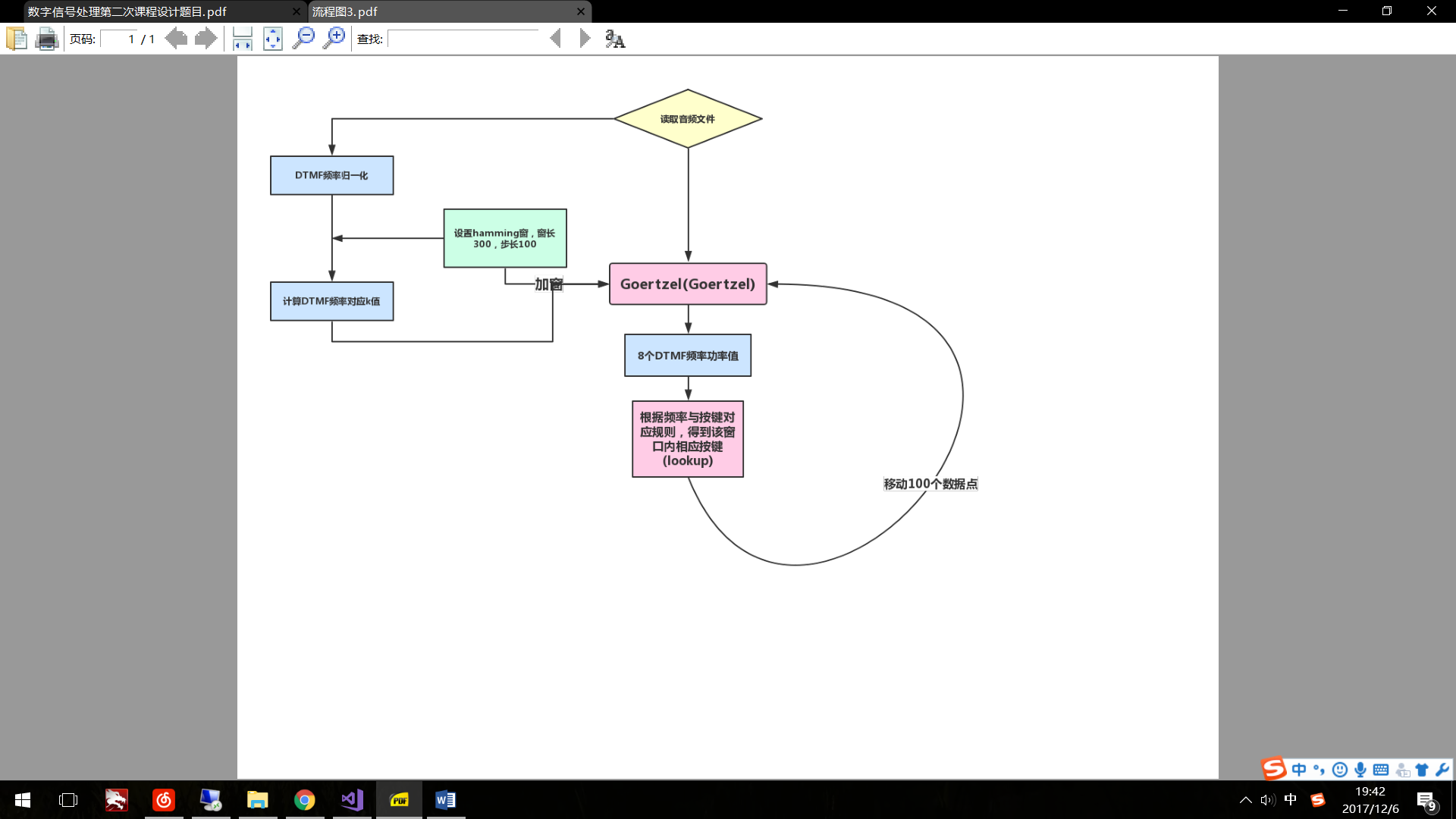
同样先将8个频率697.0, 770.0, 852.0, 941.0, 1209.0, 1336.0, 1477.0, 1633.0Hz归一化到。然后计算出这8个频率点在中对应的，并每次计算该值上下两个整数值处的频率值，并比较它们的幅值大小，将幅值更大的那个值作为实际幅度值。

其余与上FFT流程相同。

具体请见Goertzel\_anaysis函数。

**（3）**下载附件包中第二小题的一个长音频文件，文件中包含了一串 DTMF 信号，每个双音多频信号之间的时间间隔不一，对本串DTMF信号进行识别。

主要流程如下（括号中为对应的c++函数名）：



由于每个双音多频信号之间的时间间隔不一，因此采用类似短时傅里叶变换（STFT）中的方法，在时域上加窗，分析每个窗口内的频率特征。

DTMF的8个频率之间的间隔在之间，因此选取窗长为300，此时频率最小分辨率为，频率分辨力足够。同时选取每次滑动的步长为100，选取窗的类型为hamming窗。

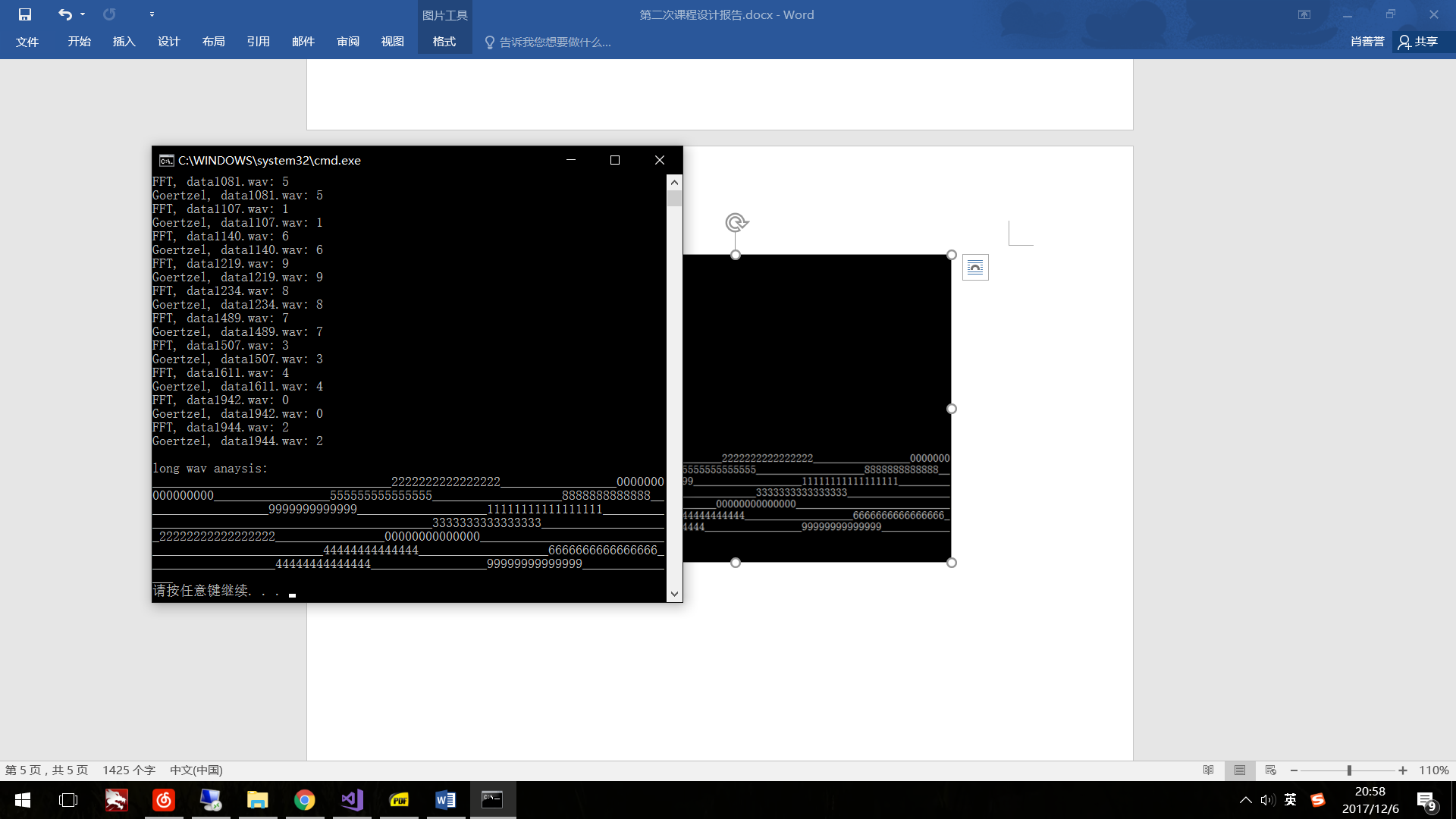
**噪声判断：**

因为每段DTMF音频之间存在空隙，因此需要设置一定的条件来判断窗口内的信号是否是有有效信号。以行频率为例：选出4个频率中最大的幅度值，并求其他3个频率处的平均值。如果**最大的幅度值 > 5倍平均值，则认为该窗口内的信号为有效信号**。如果认为不是有效信号，则输出字符 **’\_’** 。

具体请见long\_wav\_anaysis函数

1. **不同算法结果与复杂度比较**

（1）（2）FFT与Goertzel算法结果如下：

可以看到：利用FFT与Goertzel算法的结果一致。但两种方法的算法复杂度不同。

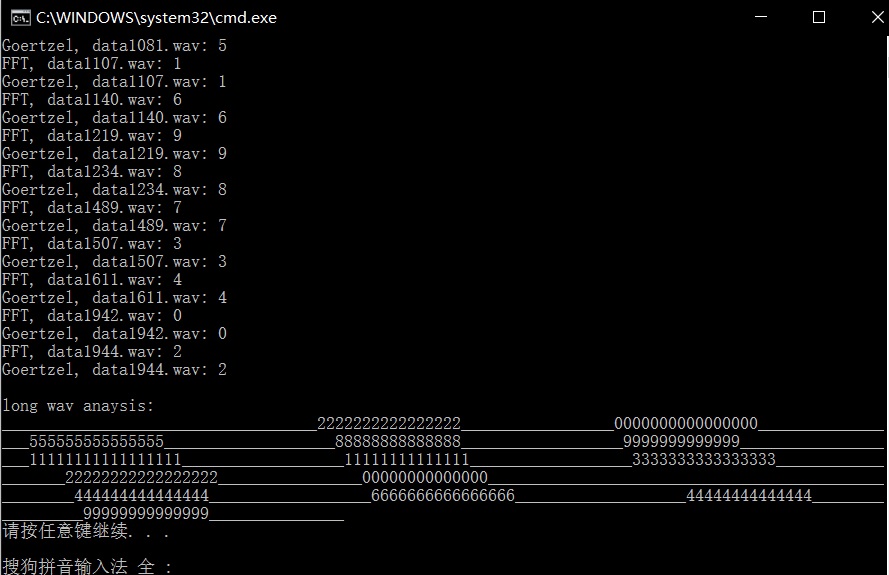
FFT需要次复数乘法次实数乘法，次复数加法，其中为向上补全至2^m的值。

而由程序设计思路中分析可知，Goertzel算法只需要实数乘法，实数加法。

相比较于直接FFT而言，在序列长度N比较大时，Goertzel算法的运算次数较少。

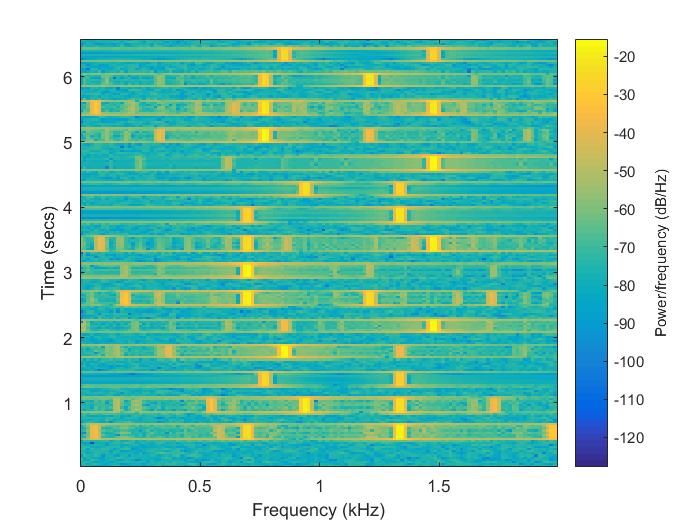
|  |  |  |  |
| --- | --- | --- | --- |
| **文件名** | **行频率** | **列频率** | **对应字符** |
| data1081.wav | 770 | 1336 | 5 |
| data1107.wav | 697 | 1209 | 1 |
| data1140.wav | 770 | 1477 | 6 |
| data1219.wav | 852 | 1477 | 9 |
| data1234.wav | 852 | 1336 | 8 |
| data1489.wav | 852 | 1209 | 7 |
| data1507.wav | 697 | 1477 | 3 |
| data1611.wav | 770 | 1209 | 4 |
| data1942.wav | 941 | 1336 | 0 |
| data1944.wav | 697 | 1336 | 2 |

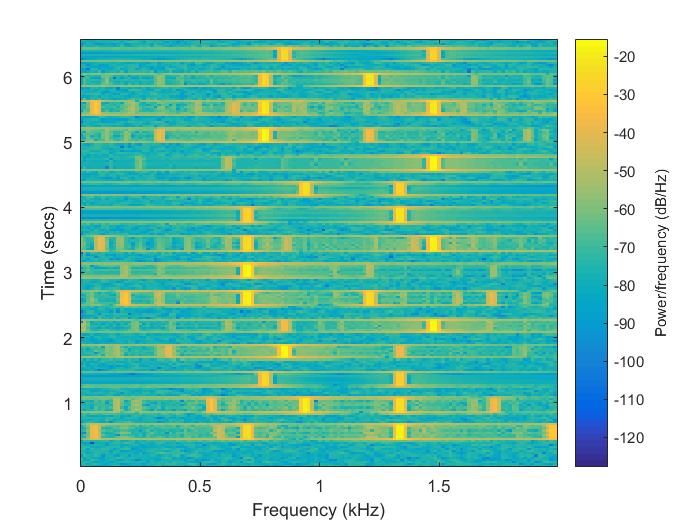
（3）利用Goertzel算法对长wav文件分析结果如下：



故实际的按键序列为20589113204649。按键之间的信号被认为是噪声信号，以字符 ‘\_’ 显示。

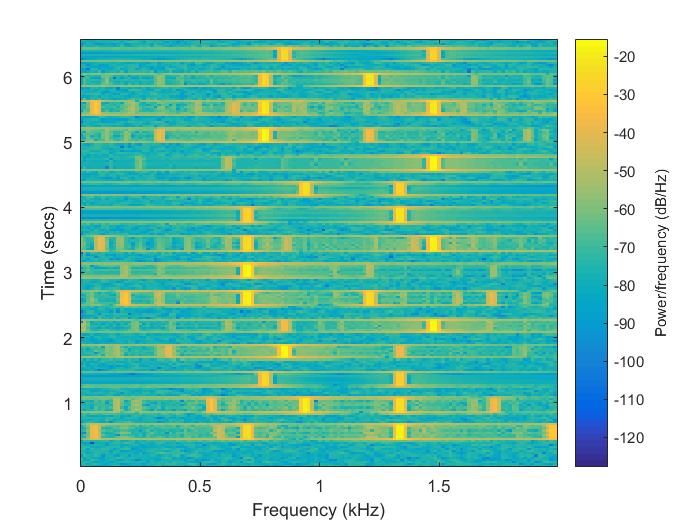
注意到整个音频中，倒数第五段信号的频率似乎有问题，似乎是一个单频信号，找不到对应的按键。长得有点像“3”，但对比整个按键序列的两个“3”的时频分析结果不完全一致，对比如下：

倒数第5段信号：

倒数第8段信号：

倒数第5段信号, 697HZ频段有些偏移，因此被程序认为是噪声信号。

利用matlab时频分析得到的时频图作为比较，结果如下：



根据时频图中的结果，查表可得，按键序列确实为2058911320(3)4649。因此利用Goertzel算法得到的结果正确。