

# 信号处理综合课程设计

## 说 明 书

设计题目： 双音多频 (DTMF) 信号的合成与识别

姓名： \_\_\_\_\_ Name 学号： \_\_\_\_\_ ID

班级： \_\_\_\_\_ Class 成绩： \_\_\_\_\_

2026 年 1 月 1 日

# 目录

<b>1 设计内容与要求</b>	<b>3</b>
1.1 设计内容	3
1.2 设计要求	3
<b>2 总体方案</b>	<b>3</b>
<b>3 方法原理</b>	<b>4</b>
3.1 DTMF 编码原理	4
3.2 Goertzel 算法	5
3.3 性能评估指标	5
<b>4 性能分析</b>	<b>5</b>
4.1 算法实现与仿真	5
4.2 时频分析	5
4.3 抗噪性能测试	7
4.4 算法对比实验	8
4.4.1 实验一：信号窗口长度与算法鲁棒性研究	8
4.4.2 实验二：自适应变积分时间检测 (Adaptive Variable Integration Time)	9
4.4.3 实验三：ESC-50 真实环境音频测试	10
4.5 理论分析	11
4.6 研究结论	11
<b>5 交互式实验演示系统实现</b>	<b>12</b>
5.1 系统架构	12
5.2 三种工作模式	12
5.2.1 实验模式 (Experiment Mode)	12
5.2.2 电话模式 (Phone Mode)	13
5.2.3 音频分析模式 (Audio Analysis)	14
5.3 关键技术实现	15
5.3.1 频率偏移模拟	15
5.3.2 ESC-50 环境噪声	15
5.3.3 WAV 存储与分析一致性	15
5.4 项目文件组织结构	15
<b>6 FPGA 硬件实现</b>	<b>16</b>
6.1 硬件架构设计	16



# 1 设计内容与要求

## 1.1 设计内容

本课程设计的主要目标是利用数字信号处理技术实现双音多频（DTMF）信号的合成及其在复杂噪声环境下的自动识别。具体内容包括：

1. **DTMF 信号合成**: 掌握 DTMF 信号的编码原则，利用 Python/Java 实现标准双频信号的生成，并具备相位连续性控制能力。
2. **核心算法实现**: 基于 Goertzel 算法实现高效的频点能量检测，替代传统的 FFT 方法，以满足实时性要求。
3. **自适应抗噪策略研究**: 针对低信噪比 ( $SNR < 0dB$ ) 场景，研究并实现基于“动态积分时长”的自适应检测算法，解决传统固定窗口算法在即时性与准确性之间的矛盾。
4. **综合性能评估**: 引入 ESC-50 真实环境噪声数据集（如雨声、风声、街道噪声），对比分析算法在非高斯、非平稳噪声下的鲁棒性。
5. **交互式系统开发**: 构建基于 B/S 架构的交互式演示系统，实现信号产生的实时可视化、噪声注入及检测过程的动态展示。

## 1.2 设计要求

1. **指标要求**: 在  $SNR = -10dB$  的高斯白噪声环境下，识别准确率需达到 95% 以上；在  $SNR = -20dB$  的极端环境下，通过自适应策略仍能保持 80% 以上的可用性。
2. **系统要求**: 演示系统需具备“实验模式”（参数调优）、“电话模式”（场景仿真）和“分析模式”（离线处理）三种功能形态。
3. **分析要求**: 深入分析 Goertzel 算法的频谱泄漏效应及相干累积增益原理，并绘制详细的性能对比曲线。
4. **工程要求**: 代码需遵循模块化设计原则，实现 Python 算法原型与 Java 工程实现的逻辑统一。

# 2 总体方案

本设计采用“合成—信道—自适应检测—交互”的闭环处理流程，总体架构如下：

1. **信号产生层**: 负责生成标准的 DTMF 信号。支持参数化控制频率偏移（模拟硬件老化）和相位初值。

2. **信道模拟层**: 构建多样化的噪声信道模型。不仅支持标准 AWGN (加性高斯白噪声)，还集成了 ESC-50 真实环境声库，模拟实际通话场景。

### 3. 自适应检测层 (核心):

- **预判级**: 使用 40ms 短窗口快速估算信噪比。
- **决策级**: 根据  $\Delta SNR = 10 \log(T_2/T_1)$  增益公式，动态计算所需的积分时长。
- **执行级**: 调用 Goertzel 算法在最佳时长下提取能量特征，结合峰值比判决输出结果。

4. **应用交互层**: 通过 Java Spring Boot 构建 Web 服务，提供可视化的波形显示、频谱分析及实时音频回放功能。

**频率设计原理**: DTMF 的 8 个频率经过精心选择，旨在最大程度减少非线性失真带来的干扰：

- **无谐波关系**: 没有任何一个频率是其他频率的整数倍，防止谐波干扰。
- **避免互调产物**: 任意两个频率的线性组合（和频与差频）都不会落在 8 个标准频率附近，有效抵抗互调失真。
- **避开工频**: 所有频率均避开电力系统的 50Hz/60Hz 及其主要谐波。

表 1: DTMF 频率编码表 (CCITT 标准)

Low / High	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

## 3 方法原理

### 3.1 DTMF 编码原理

DTMF (Dual-Tone Multi-Frequency) 信号由一个低频分量和一个高频分量叠加而成。其数学表达式为：

$$x(t) = A_L \sin(2\pi f_L t) + A_H \sin(2\pi f_H t)$$

其中  $f_L \in \{697, 770, 852, 941\}$  Hz,  $f_H \in \{1209, 1336, 1477\}$  Hz。这种双音频设计可以有效防止单一频率信号（如语言声）导致的误触发。

### 3.2 Goertzel 算法

Goertzel 算法是一种二阶递归 IIR 滤波器形式的幅度估计算法，特别适用于检测已知频点。相比于通用的 FFT 算法，它在计算少量频点时效率更高。其差分方程为：

$$s[n] = x[n] + 2 \cos\left(\frac{2\pi k}{N}\right) s[n-1] - s[n-2]$$

检测的能量值为：

$$|X(k)|^2 = s[N]^2 + s[N-1]^2 - 2 \cos\left(\frac{2\pi k}{N}\right) s[N]s[N-1]$$

### 3.3 性能评估指标

识别准确率  $P_{acc}$  定义为：

$$P_{acc} = \frac{N_{correct}}{N_{total}} \times 100\%$$

其中  $N_{correct}$  为正确识别的次数， $N_{total}$  为总测试样本数。

## 4 性能分析

### 4.1 算法实现与仿真

为了验证 DTMF 信号合成与识别算法的正确性，我们以按键”5”为例进行了仿真。

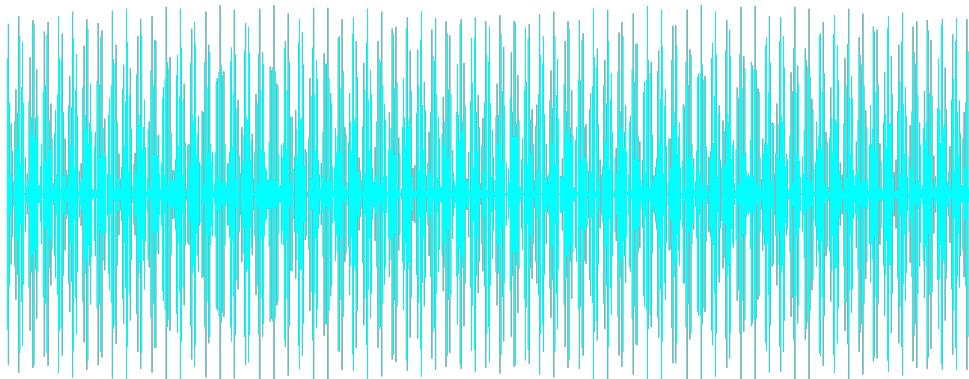


图 1: DTMF 按键”5”的时域波形图

### 4.2 时频分析

通过对合成信号进行 FFT 分析，可以看到在 770Hz 和 1336Hz 处有明显的频率分量。

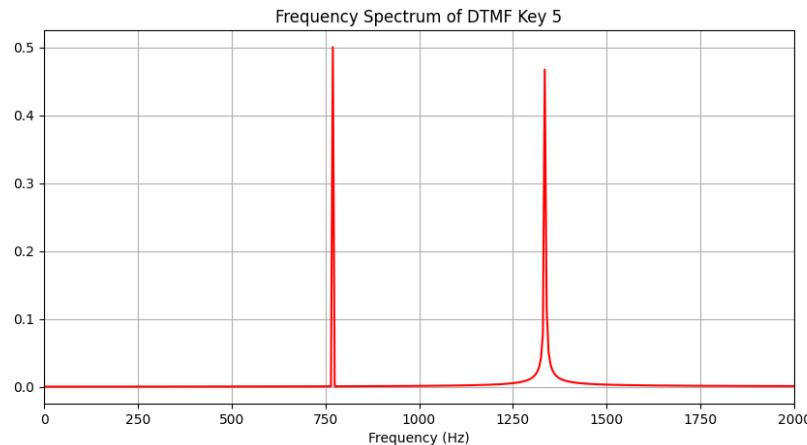


图 2: DTMF 按键”5” 的频谱图

为了更直观地展示连续拨号码时的频率特征，我们利用 FFmpeg 引擎生成了信号序列的语谱图 (Spectrogram)。

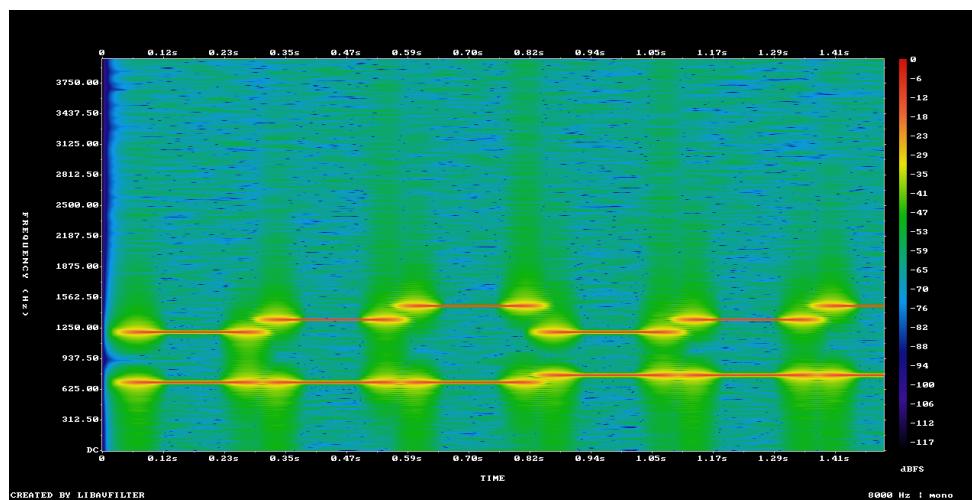


图 3: 连续拨号序列 (1-2-3-4-5-6) 的语谱图分析

利用 Goertzel 算法对 7 个目标频点进行能量检测，结果表明只有对应的两个频点能量显著升高。

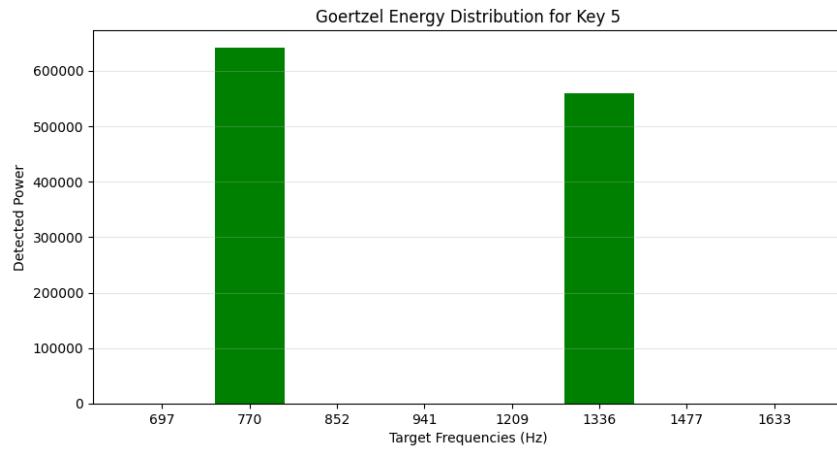


图 4: Goertzel 算法对各频点的能量检测分布

### 4.3 抗噪性能测试

在实验过程中，我们设置 SNR 从 -10dB 到 20dB 进行步进测试。

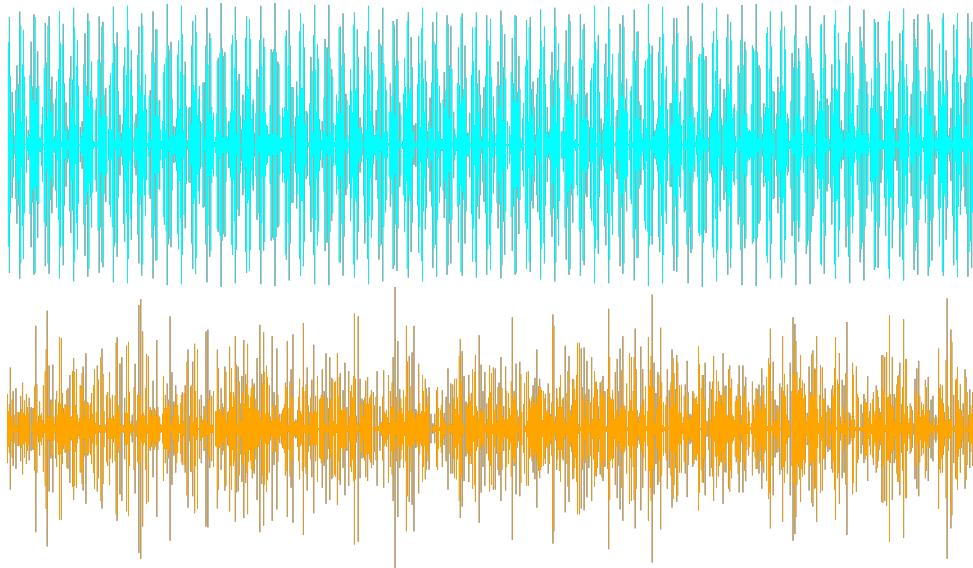


图 5: 纯净信号与  $\text{SNR} = -5\text{dB}$  噪声信号对比























从波形图中可以清晰地观察到 DDS 技术的核心原理：

- **相位累加 (锯齿波)**: 图中绿色的锯齿状波形 (signal `addr`) 代表相位累加器的高 8 位。它从 0 线性增加到 255 并循环溢出，模拟了对正弦波完整周期 ( $0 \sim 2\pi$ ) 的扫描。由于行频和列频不同，上下两组锯齿波的周期（斜率）存在明显差异，上方波形更为密集，分别对应高频（列）和低频（行）信号。
- **波形查找 (正弦波)**: 曲线波形 (signal `data`) 是根据地址从查找表 (ROM) 中读取的正弦幅度值。每个锯齿波周期严格对应一个完整的正弦波周期，验证了相位到幅度的映射逻辑完全正确，即  $\text{Output} = \sin(\text{Addr})$ 。

## 6.5 硬件测试

将比特流下载至 AX309 开发板，实际连接效果如图 16 所示。

图 16: FPGA 开发板实物连接与测试环境

## 7 附录：项目源代码

由于篇幅限制，本文档仅列出核心算法的关键实现片段。完整项目（包含 Java Web 后端、前端资源、Python 仿真及 FPGA 工程）已开源。

开源仓库地址：<https://github.com/ReragoAliNa/DTMF>

## 7.1 Python 核心算法实现 (Goertzel)

Listing 1: src/core/dsp.py (部分截取)

```
1 import numpy as np
2 from scipy import signal as scipy_signal
3 from . import config
4
5 def bandpass_filter(sig, low_freq=600, high_freq=1600, order=4):
6     """
7         带通滤波预处理，保留 DTMF 频段 (600-1600Hz)
8     """
9     nyquist = config.fs / 2
10    low = low_freq / nyquist
11    high = high_freq / nyquist
12    b, a = scipy.signal.butter(order, [low, high], btype='band')
13    filtered = scipy.signal.filtfilt(b, a, sig)
14    return filtered
15
16 def generate_dtmf(key, snr_db=None, duration=None):
17     """
18         生成 DTMF 信号
19         :param key: 按键字符
20         :param snr_db: 信噪比 (dB)，若为 None 则不加噪
21         :param duration: 信号时长 (s)，若为 None 则使用 config 默认值
22         :return: 信号数组
23     """
24     if duration is None:
25         duration = config.duration
26
27     fL, fH = config.freq_map[key]
28     t = np.linspace(0, duration, int(config.fs * duration), endpoint=False)
29     signal = np.sin(2 * np.pi * fL * t) + np.sin(2 * np.pi * fH * t)
30
31     if snr_db is not None:
32         signal_power = np.mean(signal**2)
33         snr_linear = 10***(snr_db / 10)
34         noise_power = signal_power / snr_linear
35         noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
36         signal = signal + noise
37
38     return signal
39
40 def goertzel(signal, target_freq):
```



```
27         when 11 => -- 'B'
28             inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <=
29             calc_phase_inc(COL_FREQS(3));
30
31         when 7 => -- '7'
32             inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
33             calc_phase_inc(COL_FREQS(0));
34         when 8 => -- '8'
35             inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
36             calc_phase_inc(COL_FREQS(1));
37         when 9 => -- '9'
38             inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
39             calc_phase_inc(COL_FREQS(2));
40         when 12 => -- 'C'
41             inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <=
42             calc_phase_inc(COL_FREQS(3));
43
44         when 14 => -- '*' (using 14)
45             inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
46             calc_phase_inc(COL_FREQS(0));
47         when 0 => -- '0'
48             inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
49             calc_phase_inc(COL_FREQS(1));
50         when 15 => -- '#' (using 15)
51             inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
52             calc_phase_inc(COL_FREQS(2));
53         when 13 => -- 'D'
54             inc_row <= calc_phase_inc(ROW_FREQS(3)); inc_col <=
55             calc_phase_inc(COL_FREQS(3));
56
57         when others =>
58             inc_row <= 0; inc_col <= 0;
59     end case;
60 end process;
61
62
63 -- Phase Accumulation
64 process(clk, rst_n)
65 begin
66     if rst_n = '0' then
67         phase_acc_row <= (others => '0');
68         phase_acc_col <= (others => '0');
69     elsif rising_edge(clk) then
70         if key_valid = '1' then
71             phase_acc_row <= phase_acc_row + to_unsigned(inc_row,
72 PH_ACC_WIDTH);
```

```
62         phase_acc_col <= phase_acc_col + to_unsigned(inc_col,
63             PH_ACC_WIDTH);
64     else
65         phase_acc_row <= (others => '0');
66         phase_acc_col <= (others => '0');
67     end if;
68 end if;
end process;
```

## 8 参考文献

### 参考文献

- [1] ITU-T Recommendation Q.23. (1988). *Technical features of push-button telephone sets*. International Telecommunication Union.
- [2] ITU-T Recommendation Q.24. (1988). *Multifrequency push-button signal reception*. International Telecommunication Union.
- [3] Goertzel, G. (1958). An Algorithm for the Evaluation of Finite Trigonometric Series. *American Mathematical Monthly*, 65(1), 34-35.
- [4] Oppenheim, A. V., & Schafer, R. W. (2010). *Discrete-Time Signal Processing* (3rd ed.). Pearson.
- [5] Proakis, J. G., & Manolakis, D. G. (2006). *Digital Signal Processing: Principles, Algorithms, and Applications* (4th ed.). Prentice Hall.