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

8.3 板级硬件测试 (Hardware Test)

最后，将生成的比特流下载至 AX309 (Spartan-6) 开发板进行实测。测试环境如图 15 所示，通过示波器捕捉引脚输出，并驱动扬声器进行听觉验证。实验结果表明，按键触发灵敏，输出音调准确，达到了设计预期。

图 15: FPGA 开发板实物连接与测试现场

9 附录：项目源代码

由于篇幅限制，仅列出核心算法的关键实现片段。

9.1 Python 核心算法实现 (Goertzel)

Listing 1: src/core/dsp.py (部分截取)

```
import numpy as np
from scipy import signal as scipy_signal
from . import config

def bandpass_filter(sig, low_freq=600, high_freq=1600, order=4):
    """
    带通滤波预处理，保留 DTMF 频段 (600-1600Hz)
    """
    nyquist = config.fs / 2
```

```
    low = low_freq / nyquist
    high = high_freq / nyquist
    b, a = scipy_signal.butter(order, [low, high], btype='band')
    filtered = scipy_signal.filtfilt(b, a, sig)
    return filtered

def generate_dtmf(key, snr_db=None, duration=None):
    """
    生成 DTMF 信号
    :param key: 按键字符
    :param snr_db: 信噪比 (dB)，若为 None 则不加噪
    :param duration: 信号时长 (s)，若为 None 则使用 config 默认值
    :return: 信号数组
    """
    if duration is None:
        duration = config.duration

    fL, fH = config.freq_map[key]
    t = np.linspace(0, duration, int(config.fs * duration), endpoint=
    False)
    signal = np.sin(2 * np.pi * fL * t) + np.sin(2 * np.pi * fH * t)

    if snr_db is not None:
        signal_power = np.mean(signal**2)
        snr_linear = 10**(snr_db / 10)
        noise_power = signal_power / snr_linear
        noise = np.random.normal(0, np.sqrt(noise_power), len(signal))
        signal = signal + noise

    return signal

def goertzel(signal, target_freq):
    """
    Goertzel 算法计算特定频率能量
    """
    N = len(signal)
    if N == 0: return 0
```

9.2 FPGA 信号发生器逻辑 (VHDL)

Listing 2: fpga/dtmf_generator.vhd (核心逻辑)

```

```

```
    -- Frequency Selection Logic
    process(key_idx)
    begin
        -- Standard DTMF Keypad Mapping
        -- 1(0,0) 2(0,1) 3(0,2) A(0,3)
        -- 4(1,0) 5(1,1) 6(1,2) B(1,3)
        -- 7(2,0) 8(2,1) 9(2,2) C(2,3)
        -- *(3,0) 0(3,1) #(3,2) D(3,3)
        -- Mapping key_idx 0..15 to Row/Col indices
        case key_idx is
            when 1 =>  -- '1'
                inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <= calc_phase_inc(COL_FREQS(0));
            when 2 =>  -- '2'
                inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <= calc_phase_inc(COL_FREQS(1));
            when 3 =>  -- '3'
                inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <= calc_phase_inc(COL_FREQS(2));
            when 10 => -- 'A' (using index 10 for A)
                inc_row <= calc_phase_inc(ROW_FREQS(0)); inc_col <= calc_phase_inc(COL_FREQS(3));

            when 4 =>  -- '4'
                inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <= calc_phase_inc(COL_FREQS(0));
            when 5 =>  -- '5'
                inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <= calc_phase_inc(COL_FREQS(1));
            when 6 =>  -- '6'
                inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <= calc_phase_inc(COL_FREQS(2));
            when 11 => -- 'B'
                inc_row <= calc_phase_inc(ROW_FREQS(1)); inc_col <= calc_phase_inc(COL_FREQS(3));

            when 7 =>  -- '7'
                inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <= calc_phase_inc(COL_FREQS(0));
            when 8 =>  -- '8'
                inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <= calc_phase_inc(COL_FREQS(1));
            when 9 =>  -- '9'
                inc_row <= calc_phase_inc(ROW_FREQS(2)); inc_col <= calc_phase_inc(COL_FREQS(2));
```

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

10 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.