基于 Artix7 的音乐可视化氛围灯

第一部分　设计概述

1.1 设计目的

随着当下人们生活水平的提高，大众对精神生活的需求越来越大，音乐对大众生活的影响与日俱增。因此，对于音频信息的处理、响应、呈现就显得格外重要，极具市场价值。目前市面上出现的氛围灯基本都是设定好的变化规律，不会随着音乐节奏变化甚至没有变化的。我们在这个背景下，决定设计一个能够实时随音乐而颜色亮度都随之变化的氛围灯，从而达到所听即所见的效果。

1.2 应用领域

随着人民生活水平的不断提高，人们希望音乐的表现形式不再局限于仅仅为听到，并且要求其他的表现形式实时性好，该设计能够将声音特征转换为灯光变化，并且响应速度快，采用多种算法使其更接近人的感官体验，可应用于 KTV，演唱会，高档饭店的氛围灯光，也可以作为家庭娱乐的氛围灯光，本设计亦可以应用于改造预定顺序的音乐喷泉等传统音乐表现形式，使其更富有表现力。

1.3 主要技术特点

使用 Mel 滤波器组，使功率谱根据人耳的非线性进行滤波，使其更接近人耳的主观感受。使用伽马校正，使灯光根据人眼的非线性进行校正，使人对灯光的感受近乎于线性。数字信号处理和灯带驱动均在 FPGA 内部完成，功耗低，速度快。相较于 CPU 进行处理，本设计的处理速度更快，延时更低，实时性更好。

1.4 关键性能指标

显示延迟	约$16\mu s$($100MHz$,约16000个时钟周期)
刷新频率	$90Hz$
信号处理功耗	约$0.148W$
采样速率	$20KHz$
灯带灯珠数量	76个

灯带功率	最大$22.8W$
灯带色彩	$16777216(2^{24})$种

表 1 硬件性能指标

Resource	Utilization	Available	Utilization %
LUT	4621	20800	22.22
LUTRAM	1060	9600	11.04
FF	6104	41600	14.67
BRAM	5.50	50	11.00
DSP	24	90	26.67
IO	5	210	2.38

表 2 FPGA 资源占用

1.5 主要创新点

（1） 将音乐的特征转换为灯带灯光的变化，实现所听即所见的效果。

（2） 使用 Mel 滤波器组，使功率谱根据人耳的非线性进行滤波，使其更接近人耳的主观感受。使用伽马校正，使灯光根据人眼的非线性进行校正，使人对灯光的感受近乎于线性。

（3） 使用 FPGA 实现了预加重，功率谱，Mel 滤波，时域指数滤波，伽马校正等多种算法。

（4） 所有的信号处理均在 FPGA 内实现，相较于使用 CPU 实现，本设计的处理速度更快，实时性更好。

第二部分 系统组成及功能说明

2.1 整体介绍

本作品和直接从环境中采集声音，通过对采集的声音的一系列处理，从而实现音乐可视化，达到所听即所见的效果。系统主要由麦克风模块，模数转换模块，信号处理模块（包含预加重，分帧，汉明窗，功率谱，Mel 滤波，可视化，伽马校正），灯带驱动模块（包括帧发送，比特发送）和灯带显示模块组成。麦克风采集的声音模拟信号进入 FPGA 芯片中后，由 FPGA 内置的 ADC 转换为数字信号，通过预加重，分帧，汉明窗，功率谱，Mel 滤波以匹配人的非线性听觉从而得到与人感受最相近的音频特征，随后通过可视

化模块中时域的指数滤波器组，得到与时间相关的 RGB 值，之后进行伽马校正从而匹配人对明暗感受的非线性，随后进入灯带驱动，将新的 RGB 值送至灯带。

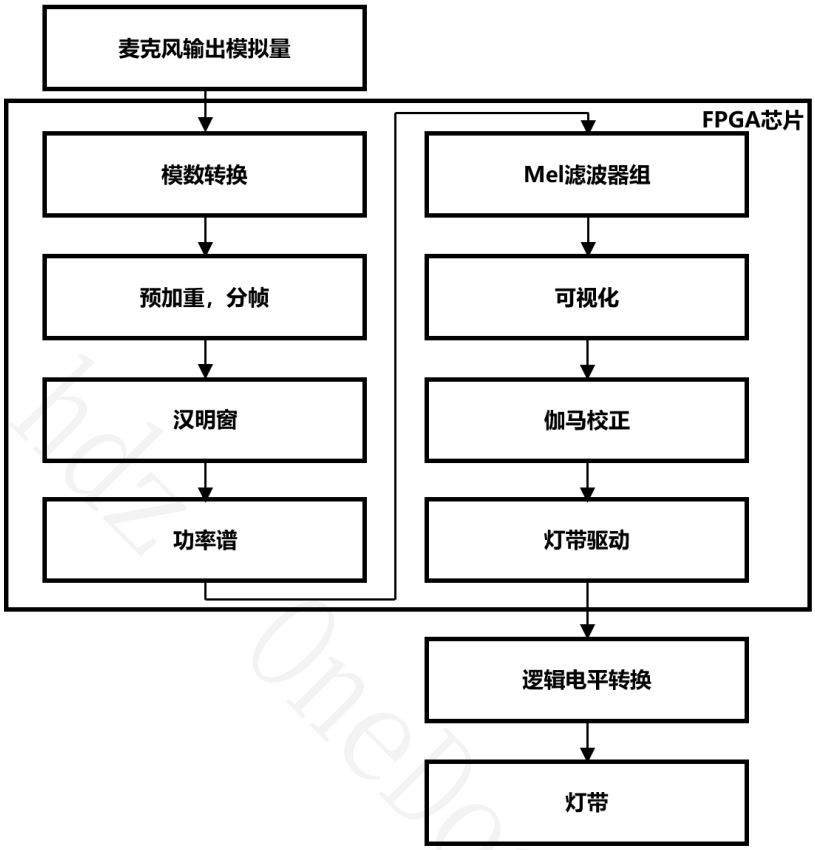

图 1 系统功能框图

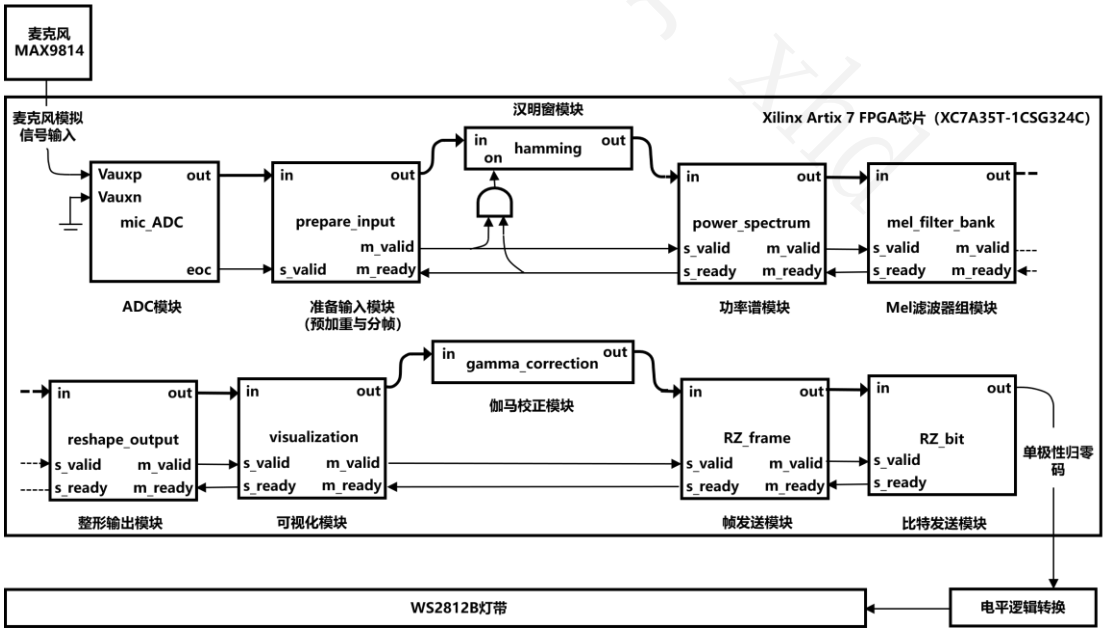

图 2 系统电路框图

2.2 各模块介绍

2.2.1 FPGA 内部模块

FPGA 内部的模块均使用简易的$AXI-Stream$协议。当主模块$valid$为高电平时并且从模块的$ready$为高电平时开始数据传输，部分模块设置有$last$信号，当传送最后一位时，$last$为高电平。

（1） ADC 模块

ADC 模块使用 FPGA 内置的 XADC。此处采用单通道模式，采样速率设置为$40KHz$（由于 XADC 能设置的最小的采样速率为$39KHz$）分频得到$20KHz$的采样信号。

（2） 预加重与分帧

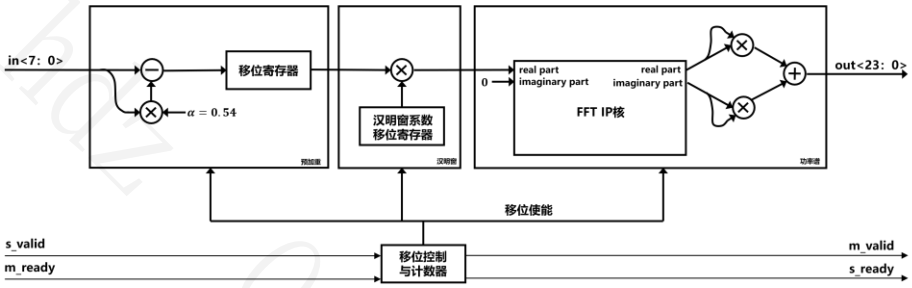

图 3 准备输入，汉明窗，功率谱模块电路框图

预加重是一种在发送端事先对发送信号的高频分量进行补偿的方法。比如对于一个 00111 的序列来说，做完预加重后序列里第一个 1 的幅度会比第二个和第三个 1 的幅度大。由于跳变 bit 代表了信号里的高频分量，所以这种方法有助于提高发送信号里的高频分量。在实际实现时，有时并不是增加跳变 bit 的幅度，而是相应减小非跳变 bit 的幅度，这种方法有时又叫去加重[1]。

我们通过采用预加重的方法对语音的高频部分进行加重，增加音乐的高频分辨率使信号的频谱变得平坦，保持在低频到高频的整个频带中，能用同样的信噪比求频谱。本设计的预加重模块采用一个 FIR 高通滤波器，该滤波器的差分方程如下所示，其中$\alpha = 0.54$。

$$y[n] = x[n] - \alpha x[n-1]$$

该滤波器的频率响应如下图所示，可以看到，该滤波器能够有效增益高频部分而抑制低频部分，从而达到预加重的效果。

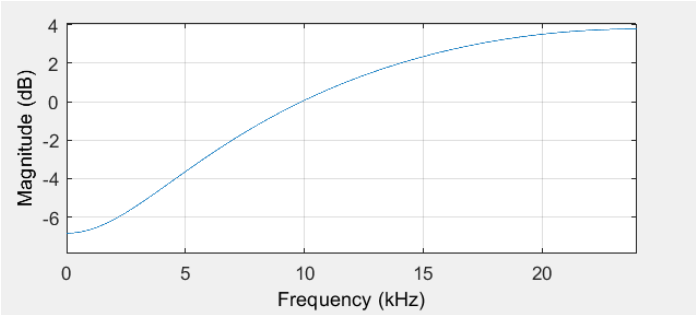

图 4 高通滤波器频率响应

每一帧的长度为512（即25.6ms），并且有长度为256（即12.8ms）的前一帧的重叠部分，即移窗1/2。通过计数器控制位移寄存器来分帧[2]。

（3） 汉明窗

加窗的目的是让一帧信号的幅度在两端渐变到 0。渐变对傅里叶变换有好处，可以让频谱上的各个峰更细，减轻频谱泄漏，这样可以降低傅里叶变换后旁瓣的强度，取得更高质量的频谱。使用汉明窗加以平滑的话，相比于矩形窗函数，会减弱 FFT 以后旁瓣大小以及频谱泄露。

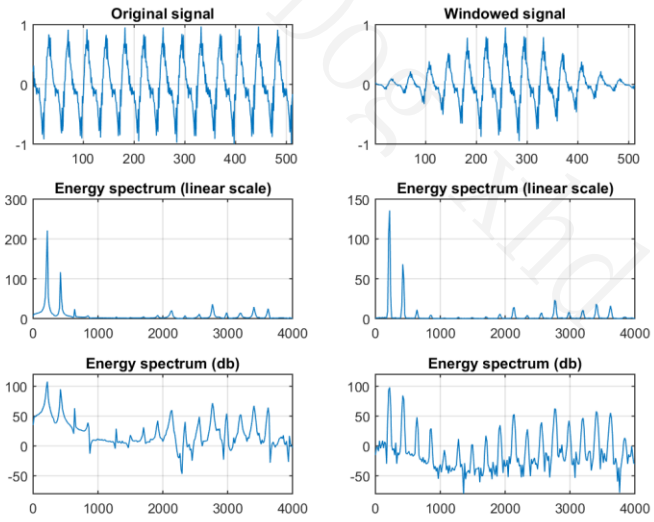

图 5 加窗信号的频谱对比

但是，代价是一帧信号两端的部分被削弱了，没有像中央的部分那样得到重视。弥补的办法是，在取帧时取相互重叠一部分。相邻两帧的起始位置的时间差叫做帧移，常见的取法是取为帧长的一半，或者固定取为 10 毫秒，本设计中取帧移为帧长的一半，即长度为256

（$12.8ms$）的帧移[3]。

原信号和加汉明窗后的信号的时域和频域的对比如图 5 所示。

加窗的过程如下，其中$a_0 = 0.54$。

$$w(n) = a_0 - (1 - a_0)cos(\frac{2\pi n}{N-1})$$

$$y(n) = x(n) \times w(n)$$

在本设计中，$w(n)$作为系数直接储存在一个位移寄存器中。下图为$a_0 = 0.54$时加汉明窗前后的时域信号。

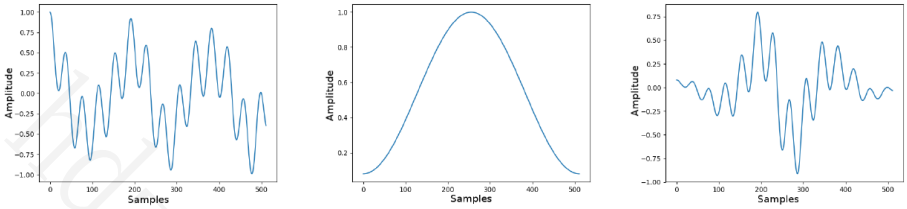

图 6 加窗信号的时域波形

（4） 功率谱

$$X = FFT(x)$$

$$P(f) = \frac{X_r(f)^2 + X_i(f)^2}{N}$$

功率谱的计算方法如上所示。计算功率谱的第一步是做 FFT，本设计中使用 Vivado 自带的 FFT IP 核，这里设置实部输入为加窗后的信号，虚部设置为零。将输出的实部虚部分别平方后求和得到幅值的平方，即总功率。由于输出是对称的，在本设计中输出长度为一半时将 FFT IP 核复位[4]。

（5） Mel 滤波器组

对于人类听觉感知的实验表明，人类听觉的感知只聚焦在某些特定的区域，而不是整个频谱包络。

而 Mel 频率分析就是基于人类听觉感知实验的。实验观测发现人耳就像一个滤波器组一样，它只关注某些特定的频率分量（人的听觉对频率是有选择性的）。也就说，它只让某些频率的信号通过，而直接无视它不想感知的某些频率信号。但是这些滤波器在频率坐标轴上却不是统一分布的，在低频区域有很多的滤波器，他们分布比较密集，但在高频区域，滤波器的数目就变得比较少，分布很稀疏。人的听觉

系统是一个特殊的非线性系统，它响应不同频率信号的灵敏度是不同的[5]。

因此，Mel 特征频率是一个新的量度，相比频率量度，梅尔更接近人耳的听觉机理。Mel 特征频率和频率的关系如下所示。

$$m = 2595log_{10}(1 + \frac{f}{700})$$

$$f = 700(10^{\frac{m}{2595}} - 1)$$

可以看到，Mel 特征频率与频率呈对数关系，如下图所示。

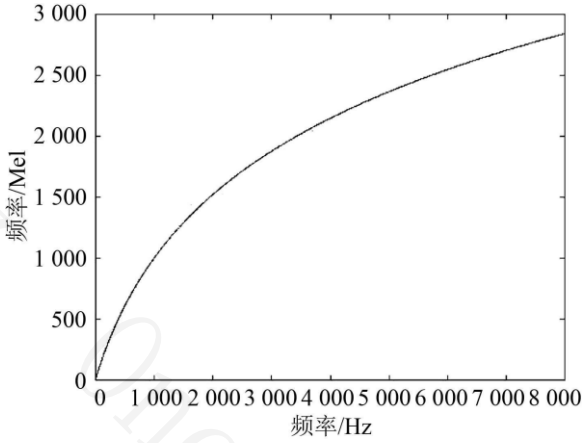

图 7 Mel 特征频谱与频率的关系

从图中可以看到，Mel 特征频率对低频敏感而对高频不敏感。

使用等高的三角形滤波器组滤波可以得到等间隔 Mel 频率特征的频谱图，Mel 滤波器组由以下公式描述。

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) < k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

滤波器的形状如下图所示，从图形上看每个滤波器的截止频率为相邻滤波器的中心频率，并且相邻的两个滤波器的增益是互补的。过 Mel 滤波器后，得到的 Mel 特征频谱是线性的。

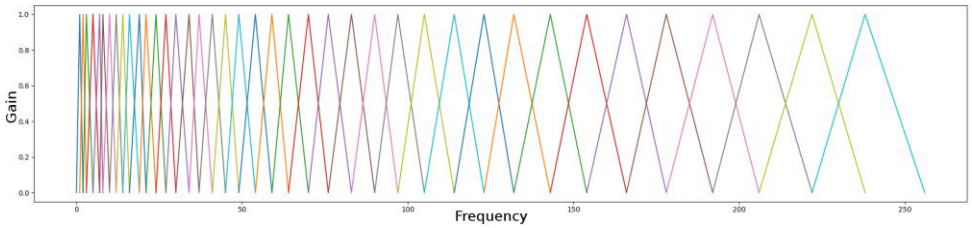

图 8 Mel 滤波器组

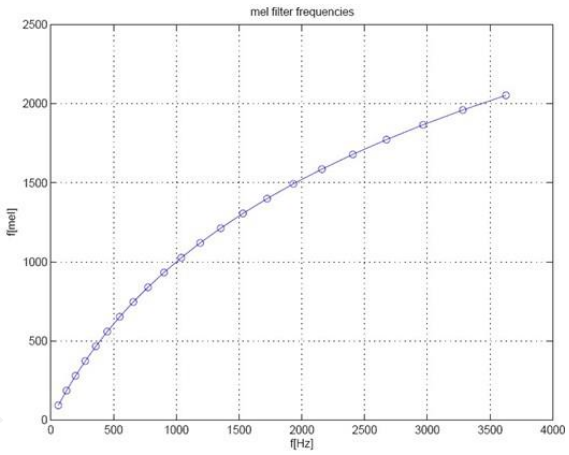

图 9 滤波器组中心频率位置

在本设计中，由于相邻的两个滤波器的增益是互补的，并且波器的截止频率为相邻滤波器的中心频率，因此采用如图所示的锯齿波状的增益，只包含各个滤波器的上升沿部分，在计算 Mel 特征频谱时通过互补同时计算相邻的两个滤波器的上升沿与下降沿。具体步骤如下（实际电路框图见图 11）：

1. 输入与增益即Mel滤波器系数移位寄存器(图 11)中的系数相乘，得到上升沿的对滤波器贡献，通过加法器不断累加，直到增益为 1，即到达中心频率，开始计算下降沿，并且将上升沿累加清零，并将上升沿的累加送入下降沿的累加。
2. 通过输入与增益即 Mel 滤波器移位寄存器中的系数相乘，再与输入相减的得到互补值，送入下降沿累加器累加，直到增益为 1，该滤波器计算结束，送入结果的位移寄存器，即 Mel 特征移位寄存器。

以上步骤是同时进行的，也就是说同时计算两个滤波器的一个上升沿和一个下降沿。

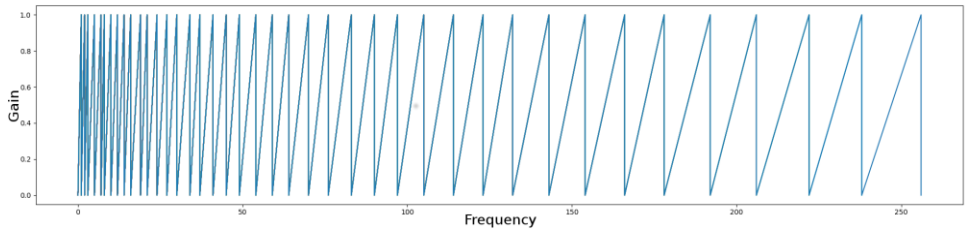

图 10 Mel 滤波器系数移位寄存器值

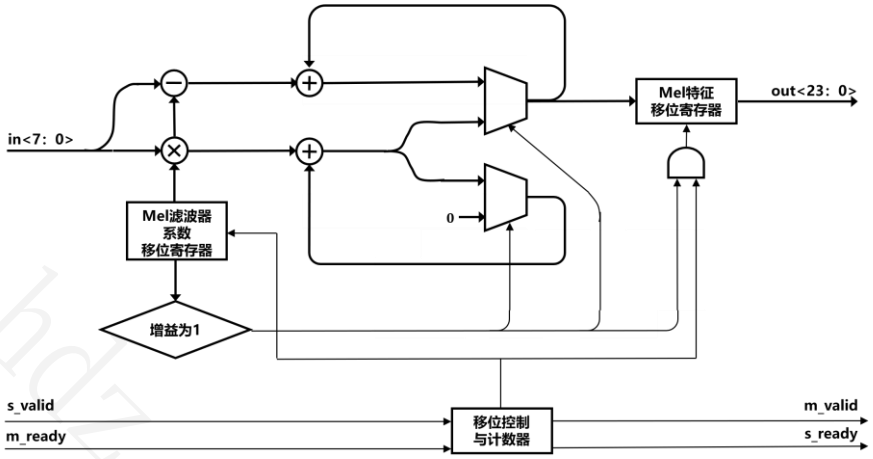

图 11 Mel 滤波器电路框图

（6） 可视化模块

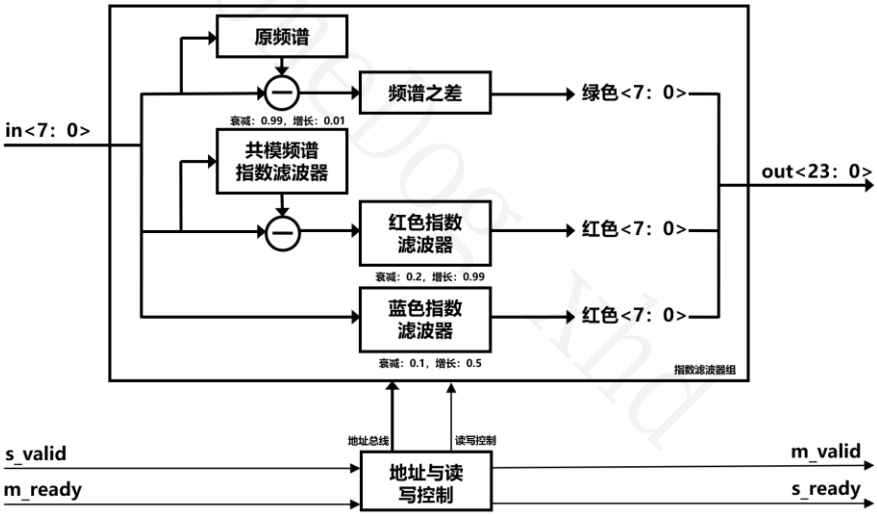

图 12 可视化模块电路框图

由于单个 Mel 频谱只有幅值的变换，可视化模块采用时域指数滤波器组对 Mel 频谱进行滤波，一方面将灯带的颜色与前后的频谱联系起来，另一方面也使灯带的变化趋于平缓，每种颜色的滤波方法均不相同，从而使灯带的颜色有了变化。各颜色的滤波方法和滤波器参数见图。

（7） 伽马校正模块

由于人眼的非线性，即亮度上的线性变化在人眼看来是非均匀的，通俗来讲，从 0 亮度变到 0.01 亮度，人眼是可以察觉到的，但从 0.99 变到 1.0，人眼可能就根本差别不出来，觉得它们是一个颜色。也就是说，人眼对暗部的变化更加敏感，而对亮部变化其实不是很敏感。也就是说，人眼认为的中灰其实不在亮度为 0.5 的地方，而是在大约亮度为 0.18 的地方。此处取$\gamma = 0.5$来校正灯带亮度，使人眼看到的亮度是线性变化的。下式为伽马校正的公式。

$$V_{out} = AV_{in}^{\gamma}$$

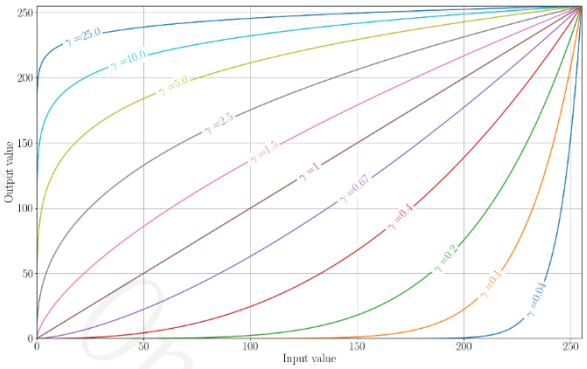

图 13 不同伽马值的校正情况

（8） 灯带驱动模块

灯带驱动模块能最小以帧单位发送数据，具有发送完成自动复位的功能。

2.2.2 外部硬件模块

（1）MAX9814 麦克风

MAX9814 是一款低成本、高性能麦克风放大器，具有自动增益控制(AGC)和低噪声麦克风偏置。器件具有低噪声前端放大器、可变增益放大器(VGA)、输出放大器、麦克风偏置电压发生器和 AGC 控制电路。

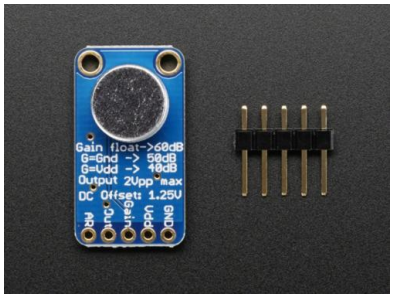

图 14 MAX9814 实物图

（2）WS2812 全真色彩灯带

WS2812 全真色彩灯带的数据协议采用单线归零码的通讯方式，像素点在上电复位以后，DIN 端接受从控制器传输过来的数据，首先送过来的 24bit 数据被第一个像素点提取后，送到像素点内部的数据锁存器，剩余的数据经过内部整形处理电路整形放大后通过 DO 端口开始转发输出给下一个级联的像素点，每经过一个像素点的传输，信号减少 24bit。像素点采用自动整形转发技术，使得该像素点的级联个数不受信号传送的限制，仅仅受限信号传输速度要求。

$TH0$	0码高电平时间	$0.3\mu s \pm 0.15\mu s$	$T0/T1$
$TL0$	0码低电平时间	$0.95\mu s \pm 0.15\mu s$	0/1码码元长度
$TH1$	1码高电平时间	$0.3\mu s \pm 0.15\mu s$	$1.25\mu s \pm 0.3\mu s$
$TH0$	1码低电平时间	$0.3\mu s \pm 0.15\mu s$	
$RESET$	帧单位，低电平	$300\mu s$以上	

表 3 WS2812B 码元定义

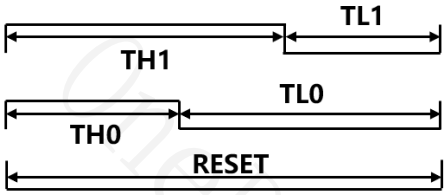

图 15 输入码型

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0

图 16 帧结构

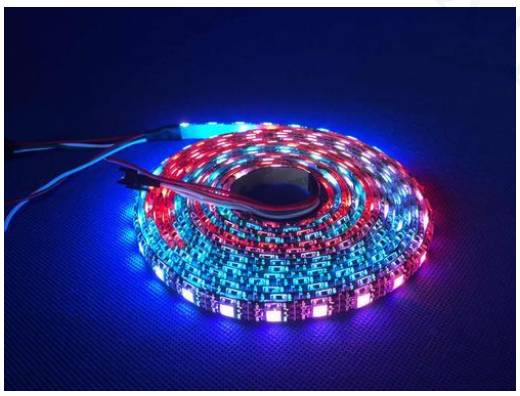

图 17 WS2812B 实物图

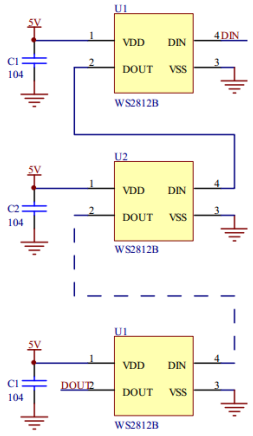

图 18 WS2812B 级联图

（2）EGO1 开发板

EGO1 是依元素科技基于 Xilinx Artix 7 FPGA 研发的便携式数模混合 基础教学平台。EGO1 配备的 FPGA (XC7A35T 1CSG324C)具有大容量高性能等特点，能实现较复杂的数字逻辑设计，平台拥有丰富的外设，以及灵活的 通用扩展接口。

EGO1 采用 Xilinx Artix 7 系列 XC7A35T 1CSG324C FPGA，其资源如下：

	Part Number	XC7A12T	XC7A15T	XC7A25T	XC7A35T
Logic Resources	Logic Cells	12,800	16,640	23,360	33,280
	Slices	2,000	2,600	3,650	5,200
	CLB Flip-Flops	16,000	20,800	29,200	41,600
Memory Resources	Maximum Distributed RAM (Kb)	171	200	313	400
	Block RAM/FIFO w/ ECC (36 Kb each)	20	25	45	50
	Total Block RAM (Kb)	720	900	1,620	1,800
Clock Resources	CMTs (1 MMCM + 1 PLL)	3	5	3	5
I/O Resources	Maximum Single-Ended I/O	150	250	150	250
	Maximum Differential I/O Pairs	72	120	72	120
Embedded Hard IP Resources	DSP Slices	40	45	80	90
	PCIe® Gen2[1]	1	1	1	1
	Analog Mixed Signal (AMS) / XADC	1	1	1	1
	Configuration AES / HMAC Blocks	1	1	1	1
	GTP Transceivers (6.6 Gb/s Max Rate)[2]	2	4	4	4
Speed Grades	Commercial	-1, -2	-1, -2	-1, -2	-1, -2
	Extended	-2L, -3	-2L, -3	-2L, -3	-2L, -3
	Industrial	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L	-1, -2, -1L

图 19 XC7A35T 内部资源

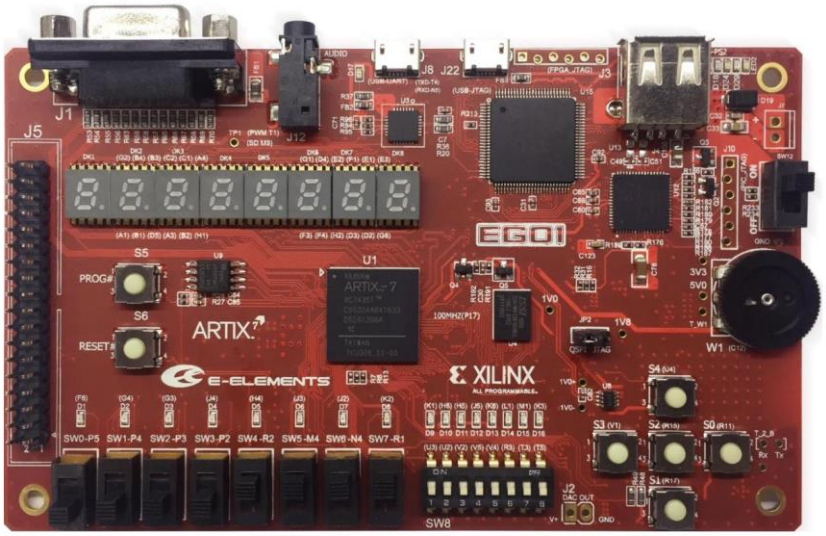

图 20 EGO1 开发板实物图

第三部分　完成情况及性能参数

将电路连接完成后，上电自动配置，可以看到灯带随声音变化而变化，无卡顿，实时性好，颜色绚丽多彩，基本实现了所听即所见的效果。

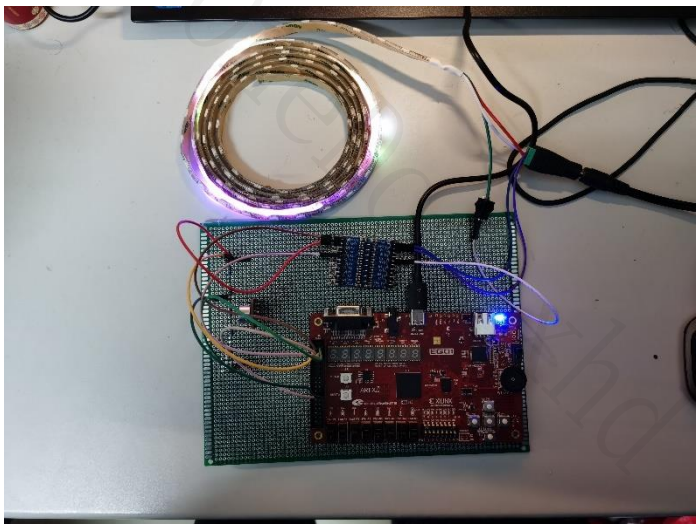

图 21 设计实物图

第四部分　附录

限于篇幅，这里仅给出顶层设计文件。

```
1.  module top(
2.      input               clk,        //时钟信号 100MHz
3.      input               rst_n,      //异步复位信号(低电平有效)
4.      input               vauxp,      //辅助差分输入(vauxp0)
5.      input               vauxn,      //辅助差分输入(vauxp1)
```

```
6.      output              RZ_data     //单极性归零码输出
7.  );
8.      wire    [11:0]      adc_out;
9.      wire                eoc;
10.
11.     wire    [15:0]      reshape_out [0:0];
12.     wire                reshape_valid;
13.     wire                reshape_last;
14.
15.     wire                visualization_valid;
16.     wire                visualization_ready;
17.
18.     wire    [23:0]      GRB_raw;
19.     wire    [23:0]      GRB;
20.
21.     wire                frame_valid;
22.     wire                frame_ready;
23.
24.     wire                RZ_bit_ready;
25.     wire                RZ_bit;
26.
27.     wire                rst;
28.
29.     reg     [15:0]      q_hamming_input;
30.     wire    [15:0]      d_hamming_input;
31.     reg     [15:0]      q_power_spec_input;
32.     wire    [15:0]      d_power_spec_input;
33.     wire    [31:0]      d_filter_bank_input;
34.     reg     [31:0]      q_filter_bank_input;
35.     wire    [15:0]      filter_bank_output  [40];
36.
37.     wire                pow_spec_ready;
38.     reg                 q_frame_valid;
39.     reg                 qq_frame_valid;
40.     wire                d_frame_valid;
41.     wire                filt_bank_ready;
42.     wire                d_pow_spec_valid;
43.     reg                 q_pow_spec_valid;
44.     wire                reshape_ready;
45.     wire                filt_bank_valid;
46.
47.     assign rst = ~rst_n;                //异步复位信号(高电平有效)
48.
49.     //FPGA 内置 ADC
```

```
50.     mic_ADC mic_ADC(
51.         .clk            (clk),
52.         .rst            (rst),
53.
54.         .out            (adc_out),
55.
56.         .eoc            (eoc),
57.
58.         .vauxp          (vauxp),
59.         .vauxn          (vauxn)
60.     );
61.
62.     //准备输入
63.     prepare_input prepare_input(
64.         .clk            (clk),
65.         .reset          (rst),
66.
67.         .in             (adc_out),
68.         .out            (d_hamming_input),
69.
70.         .s_valid        (eoc),
71.
72.         .m_ready        (pow_spec_ready),
73.         .m_valid        (d_frame_valid)
74.         );
75.
76.     //汉明窗
77.     hamming hamming(
78.         .clk            (clk),
79.         .reset          (rst),
80.
81.         .in             (q_hamming_input),
82.         .out            (d_power_spec_input),
83.
84.         .on             (q_frame_valid & pow_spec_ready)
85.         );
86.
87.     //功率谱
88.     power_spectrum power_spectrum(
89.         .clk            (clk),
90.         .reset          (rst),
91.
92.         .in             (q_power_spec_input),
93.         .out            (d_filter_bank_input),
```

```
94.
95.         .s_ready        (pow_spec_ready),
96.         .s_valid        (qq_frame_valid),
97.
98.         .m_ready        (filt_bank_ready),
99.         .m_valid        (d_pow_spec_valid)
100.         );
101.
102.     //Mel 滤波器
103.     filter_bank filter_bank(
104.         .clk            (clk),
105.         .reset          (rst),
106.
107.         .in             (q_filter_bank_input),
108.         .out            (filter_bank_output),
109.
110.         .s_ready        (filt_bank_ready),
111.         .s_valid        (q_pow_spec_valid),
112.
113.         .m_ready        (reshape_ready),
114.         .m_valid        (filt_bank_valid)
115.         );
116.
117.     //整形输出
118.     reshape_output reshape_output(
119.         .clk            (clk),
120.         .reset          (rst),
121.
122.         .in             (filter_bank_output),
123.         .out            (reshape_out),
124.
125.         .s_ready        (reshape_ready),
126.         .s_valid        (filt_bank_valid),
127.
128.         .m_ready        (visualization_ready),
129.         .m_valid        (reshape_valid),
130.         .m_last         (reshape_last)
131.         );
132.
133.     //可视化
134.     visualization visualization(
135.         .clk            (clk),
136.         .rst            (rst),
137.
```

```
        .in             (reshape_out[0][13:6]),
        .out            (GRB_raw),

        .s_last         (reshape_last),
        .s_ready        (visualization_ready),
        .s_valid        (reshape_valid),

        .m_ready        (frame_ready),
        .m_valid        (visualization_valid)

    );

    //gamma 校正
    gamma_correction gamma_correction(
        .in             (GRB_raw),
        .out            (GRB)
    );

    //帧发送
    RZ_frame RZ_frame_inst(
        .clk            (clk),
        .rst_n          (rst_n),

        .in             (GRB),
        .out            (RZ_bit),

        .s_valid        (visualization_valid),
        .s_ready        (frame_ready),

        .m_valid        (frame_valid),
        .m_ready        (RZ_bit_ready)


    );

    //字节发送
    RZ_bit  RZ_bit_inst(
        .clk            (clk),
        .rst_n          (rst_n),

        .in             (RZ_bit),
        .out            (RZ_data),

        .s_valid        (frame_valid),
```

```
        .s_ready        (RZ_bit_ready)

    );

    always @(posedge clk) begin
            begin
                    if(filt_bank_ready) begin
                            q_filter_bank_input <= d_filter_bank_input;
                            q_pow_spec_valid <= d_pow_spec_valid;
                    end
                    if(pow_spec_ready) begin
                            q_power_spec_input <= d_power_spec_input;
                            q_hamming_input <= d_hamming_input;
                    end
                    q_frame_valid <= d_frame_valid;
                    qq_frame_valid <= q_frame_valid;
            end
    end

endmodule
```

各参数生成（Python）

```python
import numpy as np

def generate_gamma_correction_table(gamma = 2):
    static = '''''
    `timescale 1ns / 1ps

    module mul_LUT_%2d(
        input [7:0] in,
        output [7:0] out
        );

        reg [7:0] _out;

        assign out = _out;

        always @(*) begin
            casex(in)
    '''
    print(static % (gamma * 100))
    for i in range(0,256):
```

```
        print('            8\'b' + bin(i)[2:].zfill(8) + ' : _out = ' + '%.0f' % (255 * (i / 255.0) ** gamma) + ';')
    static = '''''
                default: _out = %.0d;

            endcase

        end
    endmodule
    '''
    print(static % (i ** gamma))

def generate_multiply_table(factor = 0.1):
    static = '''''
    `timescale 1ns / 1ps

    module mul_LUT_%2d(
        input [7:0] in,
        output [7:0] out
        );

        reg [7:0] _out;

        assign out = _out;

        always @(*) begin
            casex(in)
    '''
    print(static % (factor * 100))
    for i in range(0,256):
        print('            8\'b' + bin(i)[2:].zfill(8) + ' : _out = ' + '%.0d' % (i * factor) + ';')
    static = '''''
                default: _out = %.0d;

            endcase

        end
    endmodule
    '''
    print(static % (i * factor))

```

```
63. def generate_hamming_shift_reg(n=512, a0=0.54):
64.     print('logic[15:0] shift_reg[N] = {', end = '')
65.     for i in range(int(n)):
66.         x = a0-(1-a0)*np.cos(2*np.pi*i/(n-1))
67.         x = round(2**14*x)
68.         print('16\'d' + str(int(x)), end = '')
69.         if(i!=n-1):
70.             print(', ', end = '')
71.         else:
72.             print('};')
73.
74. def generate_filt_bank_shift_reg(n=512, sample_rate=20000, nfilt=40):
75.     print("logic[15:0] filt_shift_reg[N] = {", end = '')
76.     mel_banks(find_mel_pts(n=n,
77.                            sample_rate=sample_rate,
78.                            nfilt=nfilt),
79.               n)
80.
81. def find_mel_pts(n=512, nfilt=40, sample_rate=20000):
82.     low_freq_mel = 0
83.     high_freq_mel = (2595 * np.log10(1 + (sample_rate / 2) / 700))
84.     mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)
85.     hz_points = (700 * (10**(mel_points / 2595) - 1))
86.     bins = np.floor((n + 1) * hz_points / sample_rate)
87.     return(bins)
88.
89. def mel_banks(bins, n):
90.     x = 0
91.     for i in range(1, len(bins)):
92.         size = int(bins[i]-bins[i-1])
93.         x = np.hstack([x, np.linspace(0, 1, size+1)[1:]])
94.
95.     for i in range(1, len(x)):
96.         z = round(2**15*x[i])
97.         print("16'd" + str(int(z)), end = '')
98.         if(i!= n/2):
99.             print(', ', end = '')
100.         else:
101.             print('};')
```