2018 年第二届全国大学生 FPGA
创新设计邀请赛

设计报告书

作品名称： 基于 FPGA 的智能语音 3D 动态显示系统

参赛学生： 沈福周 160901118

孙 册 160901119

张 伟 160901137

(团队名称)： The Youth

（2018年11月）

设计综述

裸眼 3D 作为显示技术发展的必然趋势，被认为是最有生命力且终将成为显示技术共性平台的下一代显示技术，裸眼 3D 显示关键技术的研究将决定今后显示产业的发展。本系统设计旨在为三维立体显示领域提供一种创新性的解决方案。

一般的 LED 点阵都是平面的，比如一个字就是 16*16 点阵，而光立方属于 LED 立体阵列，是在多个等间距的平面上组合成的一个三维显示系统。其最大的特点就是超炫酷的显示效果，给人一种真实的视觉冲击，带给你对三维图像最真实的感受，让我们的生活充满了美感，简直是一种艺术的享受。

本设计通过制作一个 12*12*12 像素点的光立方，使用 FPGA 高速处理能力控制不同立体位点的 LED 的亮灭和颜色变换，实现各种图像的动态显示效果。而具体的显示图案是由我们自主开发的光立方上位机进行设计的，生成每帧图像中对应 RGB LED 灯的编码值，利用 74HC245 的强驱动能力和 SM16126 级联串转并输出控制 LED，产生连续的三维动态画面。本系统采用点、线、面、体的设计思想，最终完成了可多个光立方自由拼接的电路创新设计。该系统通过语音识别模块实现人机交互，控制其显示方式和显示效果。显示方式有：白天模式、夜灯模式、呼吸灯模式；显示效果有静态三维图像显示、字体显示、炫彩图像显示、三维动画显示等。

通过多种多样的显示效果，给人一种绚丽多彩的视觉享受。而巨大的 LED 数量、空间立体的布局以及 RGB 全彩显示效果的整齐划一，都是本系统设计的巨大挑战。

关键字 光立方 3D 动态显示 RGB 语音控制 FPGA

目 录

第一部分 设计概述 /Design Introduction

1.1 设计目的

三维动态显示系统一直是国内外主要的研究对象，已经从公共信息展示等商业应用向消费类多媒体应用渗透。当前，很多光立方设计只能显示二维的画面，或者只能显示单一不可变色的画面，缺少三维的视觉效果。本系统利用 FPGA 的高速处理性能，使用 RGB 彩色 LED 灯搭建 12*12*12 的光立方显示系统，可显示 3D 全彩动态画面（1600W 种颜色变换）。同时，采用语音控制其工作，增强了系统的人机交互性能，拓宽了其应用范围。

3D LED 显示系统源于设计和构建高品质 RGB 的 LED 光立方。由 RGB LED 组成的立方体将允许用户显示更宽的色域范围。3D LED 显示系统不仅包含光立方本身，而且为动画和修改内容创建了一个友好的用户界面。

1.2 应用领域

光立方显示系统作为一种控制领域的外围器件，可在多种应用设计中得到二次开发，比如：智能家居领域（家庭、城市装饰、大型会展等）、智能教育领域（儿童几何空间启蒙教育、兴趣启发等）。同时，也是电子类专业学习者软件编程、课程设计的最佳练习方式。

1.3 适用范围

家庭、酒店等室内灯光装饰，大型会展、城市街景建设，适用于各类小游戏的显示屏、卡拉 OK 装饰灯、呼吸灯等各种灯饰。

第二部分 系统组成及功能说明 /System Construction & Function Description

2.1 系统介绍

本项目设计制作一个 12*12*12 像素点的光立方三维显示系统，采用语音识别模块、人体红外检测模块、硬件驱动电路、FPGA 主控制器等模块控制不同立体位点 LED 灯的亮灭和颜色变换，实现各种立体图像的显示。

而具体的立体显示图案是由我们自主开发的光立方上位机进行设计的，生成每帧图像中对应 RGB LED 灯的编码值，利用 74HC245 的强驱动能力和 SM16126 级联串转并输出控制 LED，产生连续的三维动态画面。我们采用点、线、面、体的设计思想，分化为三个 12*12*4 的子立方，利用 SM16126 级联的控制底板，最终完成了可以多个光立方自由拼接的电路创新设计。

该系统可在语音识别模块的控制下，实现人机交互，通过语音控制其显示方式和显示效果。

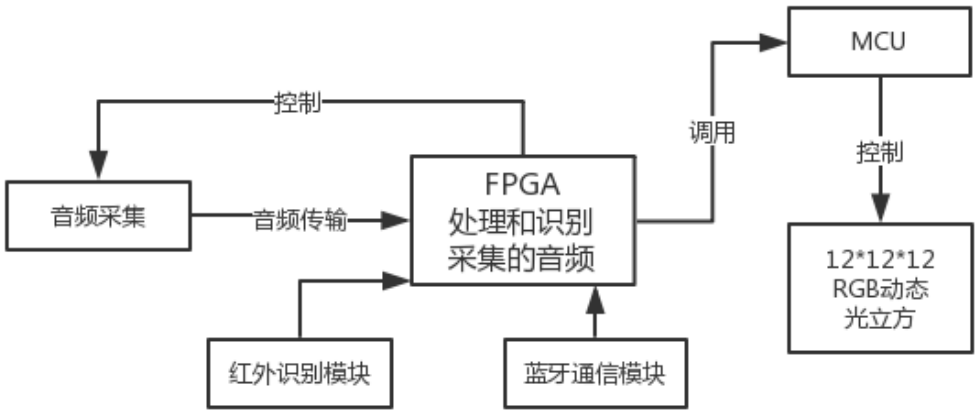


图 2.1 系统框图

2.2 各模块介绍

2.2.1 LED 灯的选用

LED 发光体的体积越小，光立方整体的通透性就越好，也就是说后排的 LED 就越不容易被前排的 LED 挡住；另一方面，发光体越大，越容易看到光

点，例如使用直径更大的 LED 或是使用屋面而非光面的 LED。此外，还要注意 LED 光点的可视角度，雾状 LED 要比光面 LED 要大。

本项目使用的 LED 灯为 5MM、RGB 雾状散光 LED 灯，其最大电流为 20mA，电压范围 3.0-3.5V，波长 460-465nm。实物如图 2.2 所示：

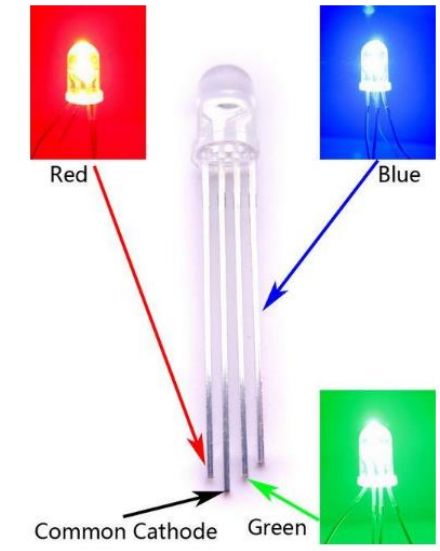


图 2.2 4 脚 RGB LED

2.2.2 蓝牙通信模块

本系统中，蓝牙模块用来实现语音识别模块与 MCU 之间的通信功能。FPGA 对采集到的音频信号进行识别处理，通过蓝牙串口发出控制指令，从机接收后 MCU 完成相应的显示效果。

图 2.3 蓝牙 HC-05 串口

2.2.3 语音识别模块

对于音频信号的处理，我们采用 LD3320 模块来进行处理。FPGA 模拟 SPI 与模块通信读取其寄存器的值，并对其做 FFT 快速傅里叶变换进行语音信号处理，通过语音信号来发送相应的指令。

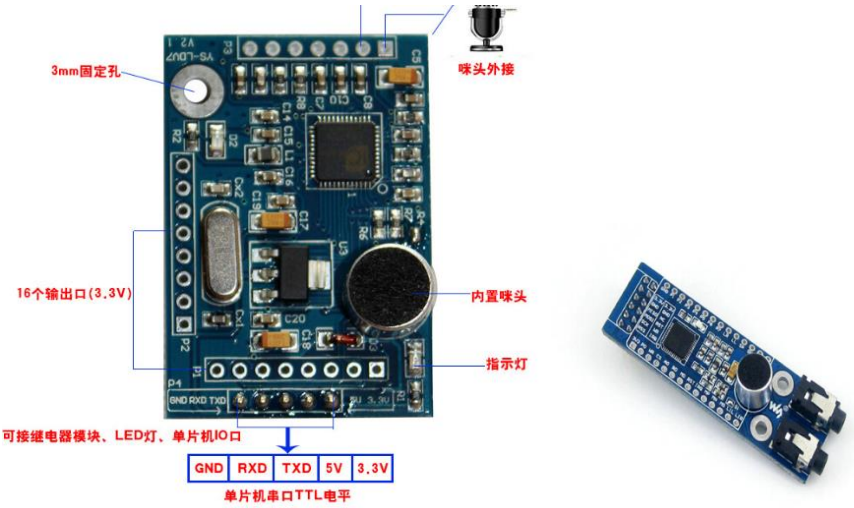


图 2.4 LD3320 语音识别模块

2.2.4 人体红外检测模块

本系统采用 HC-SR505 来实现人体检测，此模块是基于红外线技术的自动控制产品，灵敏度高，可靠性强，小体积，低电压工作模式。广泛应用于各类自动感应电器设备。

图 2.5 人体红外检测模块

2.3 硬件电路

2.3.1 SM16126 级联电路

采用 12 片 SM16126 串转并芯片（16 个输出口）级联形成 144 个输出口控制，对应于光立方的一面 LED。通过 DateIN 数据端口输入时序控制，进行 PWM 调制，产生 RBG 颜色调节。

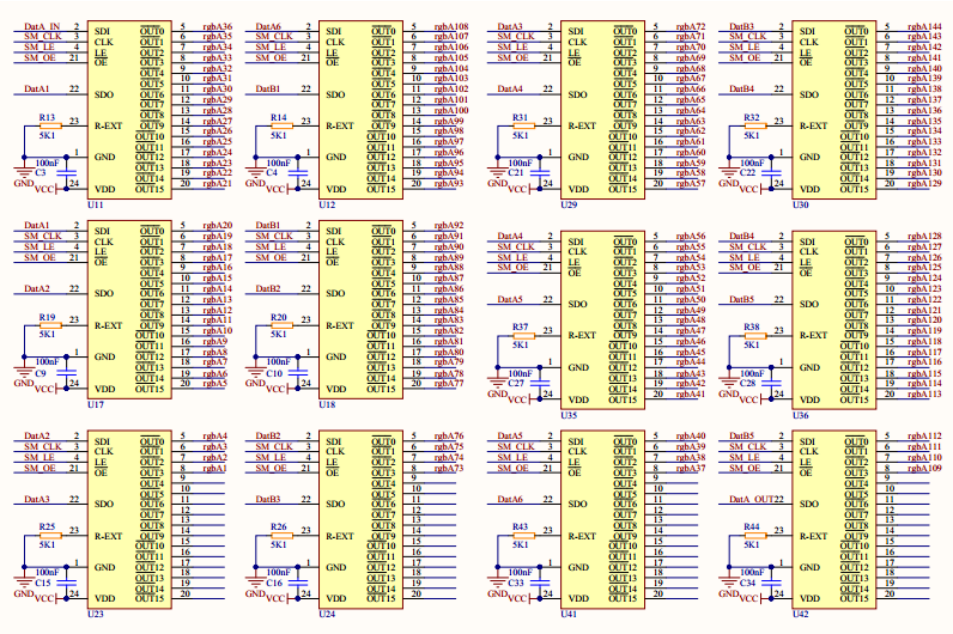


图 2.6 SM16126 级联电路

2.3.2 选层电路

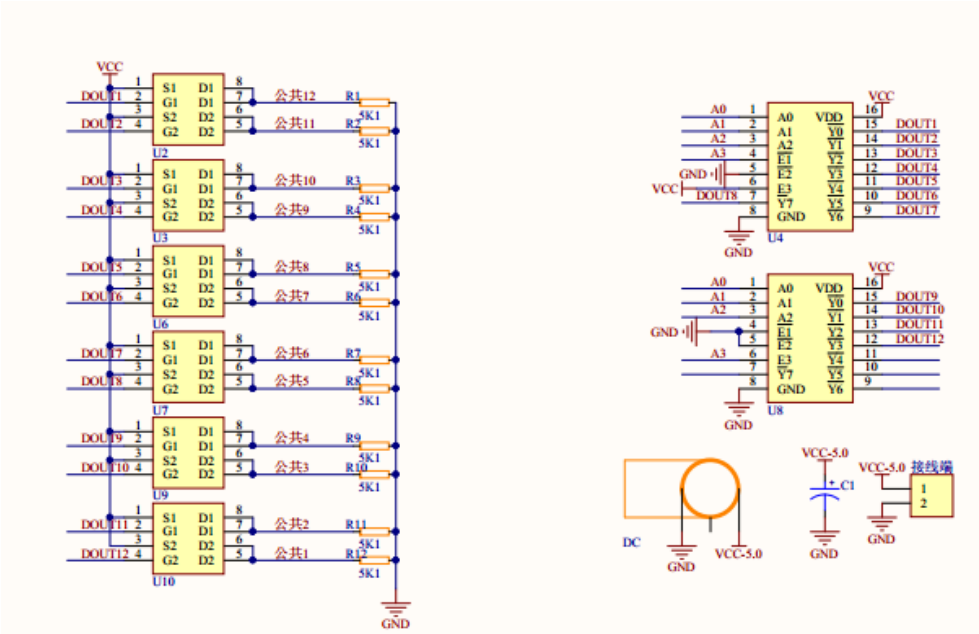


图 2.7 74HC138 选层电路原理图

2.3.2 音频输出电路

LM358 是双运算放大器。内部包括有两个独立的、高增益、内部频率补偿的运算放大器，适合于电源电压范围很宽的单电源使用，也适用于双电源工作模式，在推荐的工作条件下，电源电流与电源电压无关。它的使用范围包括传感放大器、直流增益模块和其他所有可用单电源供电的使用运算放大器的场合。本系统用其进行音频播放。

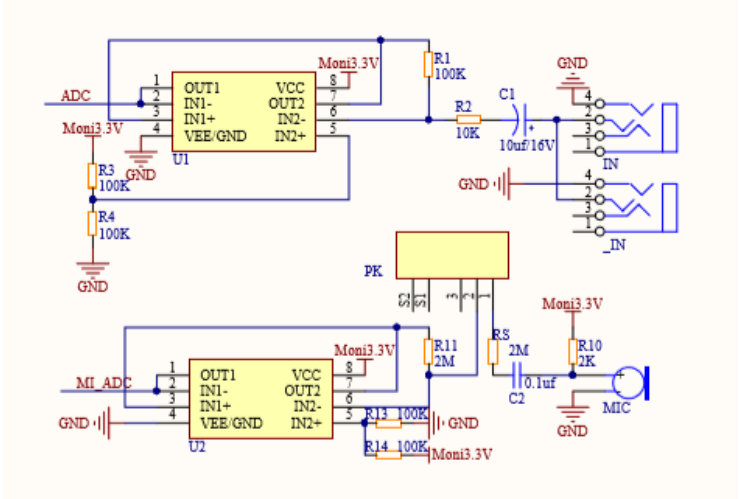


图 2.8 音频输出电路图

2.4 软件设计

2.4.1 软件简介

软件设计包括主程序、音频采集、音频处理程序、光立方显示程序、蓝牙通信程序、Fatfs 系统读取 TF 卡程序以及上位机端的图形开发程序。主程序中通过调用人体检测模块程序来检测光立方的周围是否有人，如果有就产生中断并且调用显示程序，显示开机的动画。此时语音模块开始采集周围的音频信号，当检测到相应的音频信号，经过滤波、降噪、识别的处理与语音库做对比，并产生相应的信号，通过蓝牙模块将信号发送至显示端，显示端检测收到的信号，并从 TF 卡中读入相应动画的数据，从而使得光立方显示相应的动画。同时，借助上位机端可以生成或修改用户所需要显示的动画。

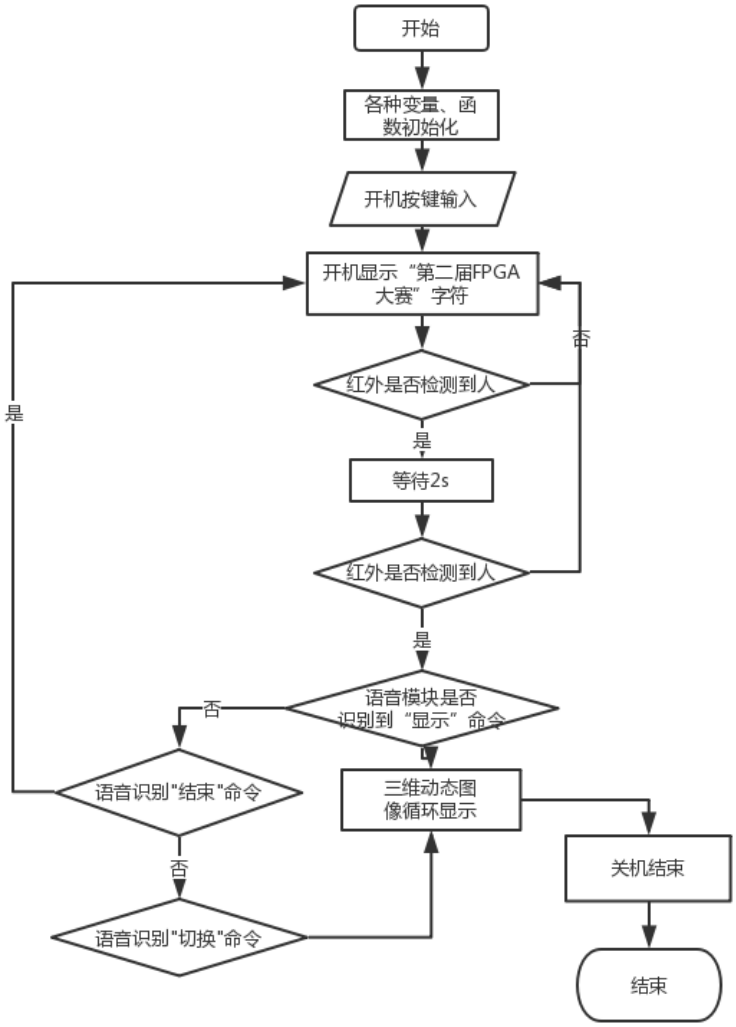


图 2.1 程序流程图

2.4.2 SDK 程序介绍

2.4.2.1 软核主函数程序

```
#include  <stdio.h>
#include  "platform.h"
#include  "xil_printf.h"
#include  "xparameters.h"
#include  "xgpio.h"
#include  "microblaze_sleep.h"
#include  "xsysmon.h"
int main(void)
{
    GPIO_init();      //GPIO 口初始化
    PeopleScan_init();   //人体检测模块初始化
```

```
    LED_CUBE_init(); //光立方显示程序初始化
    TFRead_init();       //Tf 卡读取程序初始化
    Bluetooth_init();//蓝牙通信初始化

    while(1)
    {
        while(PeopleScan_flag==0)
        {
            ReadFile(0);
            LED_CUBE_Show(0);
            //如果开机后人体检测模块没有检测到人就一直循环播放初始动画
        }
        if(Bluetooth_reveive_flag)  //蓝牙串口收到语音模块的命令数据
        {
            ReadFile(1);
            LED_CUBE_Show(1);           //显示动画 1（开机动画）
            BluetoothChart = GetBluetoothChart(); //获取接收到的数据
            switch(BluetoothChart)      //判断数据并执行相应命令
            {
                case 2:
ReadFile(2);LED_CUBE_Clear();LED_CUBE_Show(2);break;      //显示动
画 2
                case 3:
ReadFile(3);LED_CUBE_Clear();LED_CUBE_Show(3);break;      //显示动
画 3
                case 4:
ReadFile(4);LED_CUBE_Clear();LED_CUBE_Show(4);break;      //显示动
画 4
                case 5:
ReadFile(5);LED_CUBE_Clear();LED_CUBE_Show(5);break;      //显示动
画 5
                case 6:
ReadFile(6);LED_CUBE_Clear();LED_CUBE_Show(6);break;      //显示动
画 6
                case 7:
LED_CUBE_Clear();LED_CUBE_Show(1);PeopleScan_flag=0;break;   //检测
到结束信号就返回显示开机动画并且人体检测模块重新开始运行
                default:NoDo();break; //数据接收错误
            }
        }
    }
}
```

2.4.2.2 TF 读卡程序

```
//通过移植 Fatfs 系统来进行对文件的快速读写
FATFS fs;                          //Fatfs 文件系统对象
FIL fnew;                          //文件对象
UINT fnum;                         //文件成功读写的数量
FRESULT res_flash;                 //读取结果
BYTE buffer[ReadFileNum]={0};      //读缓存区
unsigned char Face[2][5184]={0};

void TFCard_Init()
{
MSD0_SPI_Configuration();                    //初始化 SPI 总线
res_flash = f_mount(0,&fs);                  //挂载文件系统
if(res_flash != FR_OK) printf("mount filesystem
faiLED : %d\n\r",res_flash);
}
void ReadFile()
{
    f_open(&fnew,Filename,FA_OPEN_EXISTING | FA_READ); //打开
文件
    f_read(&fnew,buffer,sizeof(buffer),&fnum);   //读取文件中的数据
    f_close(&fnew);            //关闭文件
    convert();          //执行数据转换
}

int convert(void)
{
    int i=0;
     static int t=0;
    unsigned char temp;
    for(i=0;i<ReadFileNum;i++)
    {
        if(buffer[i] <= 57) buffer[i]-=48;          //0~9 的数字
        else buffer[i] -= 55;               //a~f 字母
    }
    for(i=0;i<864;i+=2)                //数据排序
    {
        if(buffer[i]==15 && buffer[i+1]==15)
    {Face[0][t]=1;Face[1][t]=1;}
            else if(buffer[i]==0&&buffer[i+1]==0)
    {Face[0][t]=0;Face[1][t]=0;}
```

```
        else {Face[0][t]=1;Face[1][t]=0;}
        t++;
        if(t == 5184)
        {
            for(t=0;t<5184;t+=3)
            {
                temp=Face[0][t];
                Face[0][t]=Face[0][t+1];
                Face[0][t+1]=Face[0][t+2];
                Face[0][t+2]=temp;

                temp=Face[1][t];
                Face[1][t]=Face[1][t+1];
                Face[1][t+1]=Face[1][t+2];
                Face[1][t+2]=temp;
            }
            t=0;
        }
    }
}
```

2.4.2.3 SM16126 驱动程序

```
//通过模拟 IIC 总线来驱动 SM16126 芯片
//同时通过多次频繁的写入数据，模拟 PWM 脉宽调制来实现多灰度的显示
void WriteData(unsigned char Data_A,unsigned char Data_B,unsigned
char Data_C)
{
    SDI1 = Data_A;              //数据端口准备数据
    SDI2 = Data_B;
    SDI3 = Data_C;
    delay_us(1);
    CLK = 1;                    //时钟端口拉高，发送数据线数据
    delay_us(1);
    CLK = 0;                    //时钟端口拉低，改变数据
    delay_us(1);
}
//使能锁存器，使得数据锁存并输出
void RENStarth(void)
{
    LE = 1;                     //锁存器使能
    delay_us(1);
    LE = 0;
```

```
    delay_us(1);
    OE = 0;                    //光立方输出锁存器数据
    delay_us(1);
}
```

2.4.2.4 上位机简介

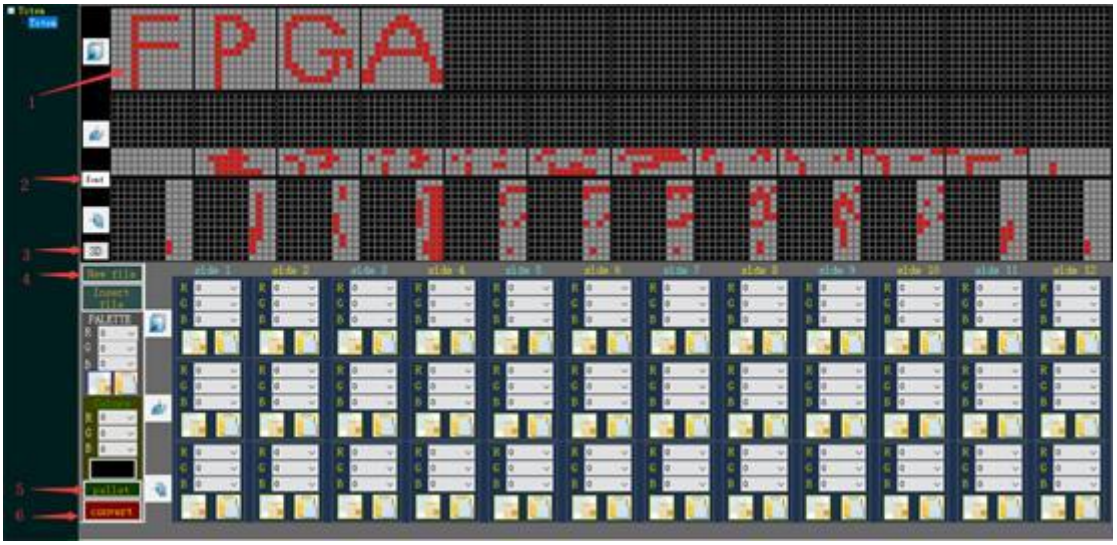


图 2.2 上位机

各部分使用说明如下：

1. 图形操作界面，便于修改需要的动画
2. 字符输入界面，可以快捷的产生各种字体的字符
3. 3D 仿真系统，便于调试所需的动画
4. 新建一个新的动画文件
5. 调色板，可以选择不同灰度的色彩
6. 将动画转换成所需的数据

第三部分 完成情况及性能参数 /Final Design & Performance Parameters

按照我们预期的规划，采用自上而下的设计方法，从系统的整体架构搭建，逐渐一个模块、一个模块地调试和完善，目前已基本完成我们的设计任务，实现了可语音控制的光立方 3D 动态画面的显示效果。

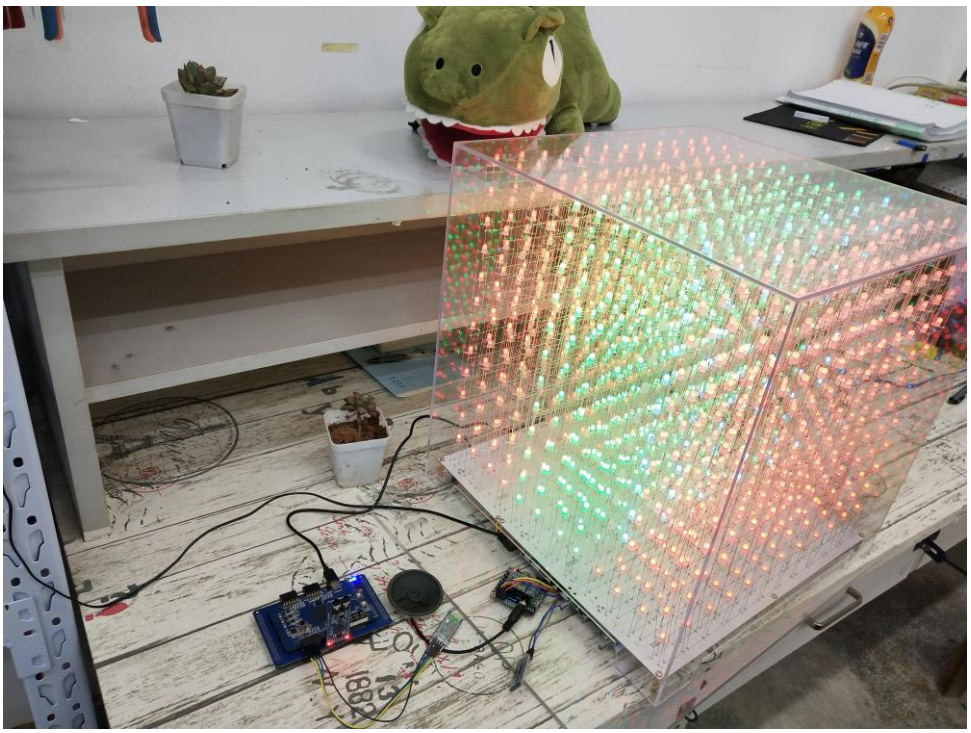


图 3.0 系统整体实物图

3.1 完成情况

3.1.1 光立方的焊接

为了保持整体的通透性、立体感，3D 光立方没有设计额外的 LED 支架，所有搭建直接使用 LED 自身的管脚。为此我们制作了模板，固定每一列、每一排 LED 以相同的间距排放，搭建出规整的立方体。

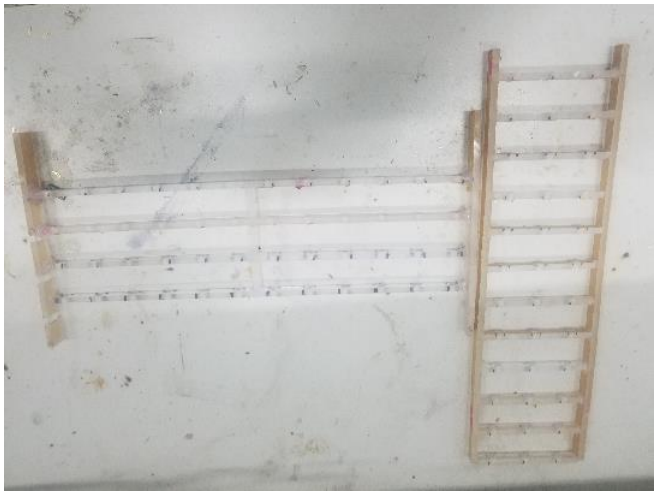


图 3.1 固定 LED 灯的模具

图 3.2 折弯的 LED 灯

我们采用点、线、面、体的思路，首先要进行 LED 灯立体阵列的搭建。

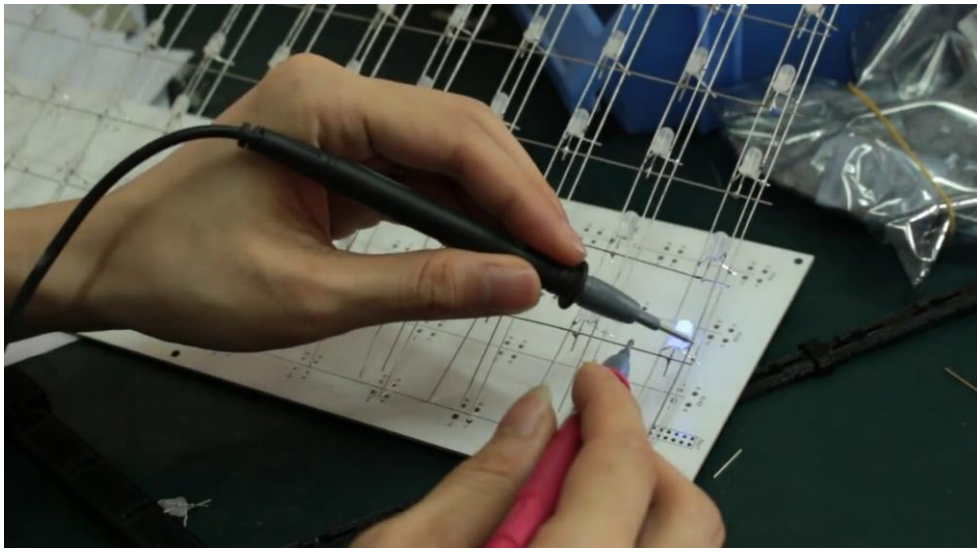


图 3.3 LED 灯条焊接及测试

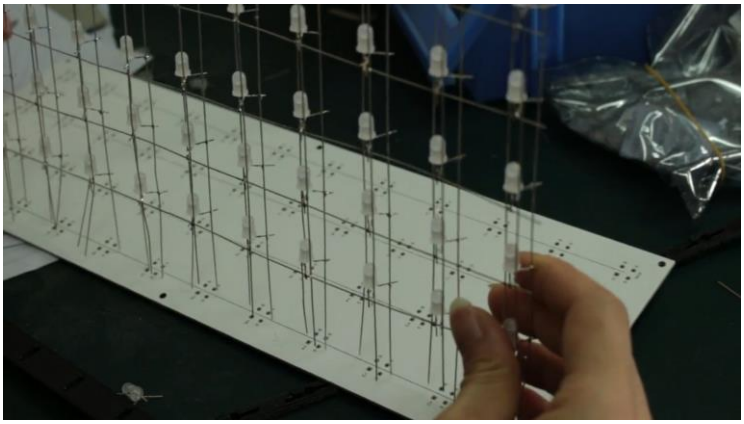


图 3.4 一面 LED 灯组装示意图

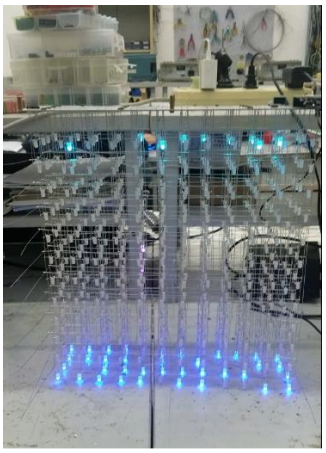


图 3.5 光立方组装测试图

图 3.6 光立方组装完成图

3.1.2 光立方的软件调试

经过各个模块的调试，并且成功组装成一个完成的系统后，可显示各种三维动态画面。

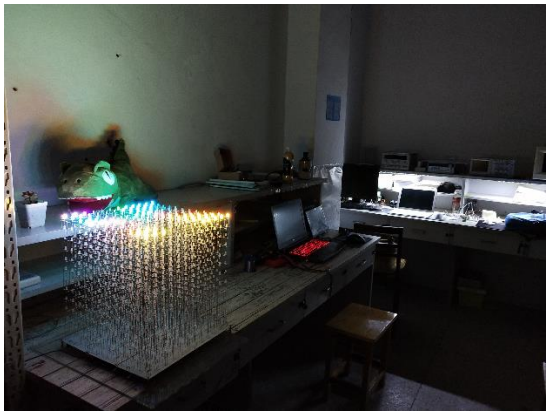


图 a “风车”

图 b “雨滴”

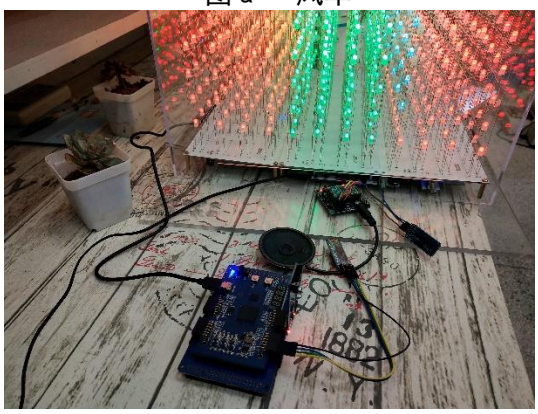


图 c 语音控制

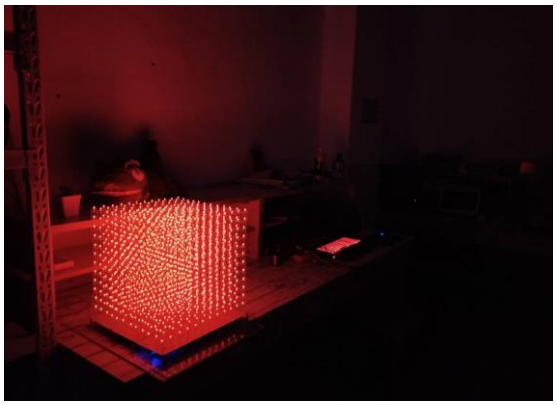


图 d 夜灯模式

3.2 性能参数

3.2.1 硬件电路模块

该系统所以 RGB LED 灯均采用共阳极，并采用 5V-2A 的电源进行供电（可以使用充电宝供电）。对于 74HC 系列芯片，我们采用开发板提供的 3.3V 稳压电源供电。

3.2.2 显示模块

该模块系统时钟为 70MHz，每一组数据写入共 16 个时钟周期，在每一个时钟信号上升沿时写入数据到芯片，在非上升沿时准备数据。写入数据阶段共需个 2 时钟周期，共 2us。对于显示时间，则需要调整。为达到 RGB 的效果，每种颜色需要设置多种亮度，而在视觉上不同的亮度与显示时间不成正比，因此需要调节显示时间，从而达到视觉上的均匀变化的亮度。

3.2.3 数据模块

一帧数据为 10864 字节，设置好一帧数据后写入 flash。

3.2.4 外壳尺寸

LED 灯横向间隔约 2.5cm，纵向间隔约 2.5cm，每一层间隔约 2.5cm，亚克力外壳整体约 38*38*40.5cm。

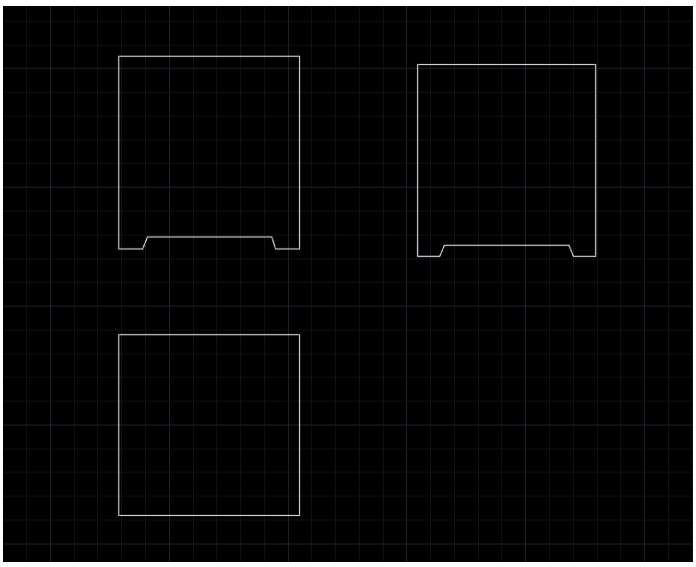


图 3.7 外壳尺寸 CAD 图

第四部分 总结 /Conclusions

4.1 主要创新点

国内外的多家公司、实验室对 3D 显示技术展开研究，目前市场上的光立方显示系统显示画面颜色较为单一，三维视觉效果不佳，控制方式单一等方面存在不足或空缺。因此，我们此系统的设计将解决此类不足，并做出创新性的提高。

1. 显示画面可自主设计

小组成员自主开发设计的上位机，用户可自主设计图案，设置显示效果，简单可操作。上位机的系统兼容性较好，可在 PC 端使用。其产生的编码可直接 load 在 TF 卡中，方便三维图案及显示效果的设计。

2. 提高人机交互性能

通过按键、红外模块、语音模块控制光立方的显示方式和显示效果。

3. 多个光立方显示系统的自由拼接

控制底板采用划归的设计思想，将 12*12*12 的 LED 灯矩阵列分划成三个 12*12*4，采用 12 片 SM16126 控制。当需要多个或者更大光立方系统时，只需将 JIN 连接起来即可实现拼接。

4. 可移动性、应用场地的多样性

本显示系统采用 5V 2A 的电源供电，电源选择面广。除了用室内的电源适配器供电外，还可使用移动电源（通常的充电宝）为其供电，实现了不受时间和地点的限制。

5. 环境友好，绿色环保

选用低功耗、发光波长较短（460~465nm）的雾状散光 LED 灯，尽量不采用大面积青蓝色图像显示，降低对人眼的伤害，提高环境友好性，节能环保。

6. 可二次开发

光立方显示系统作为一种控制领域的外围器件，可在多种应用设计中得到二次开发，比如：智能家居领域（家庭、城市装饰，大型会展等）、智能教育领域（儿童几何空间启蒙教育、兴趣启发等）。系统的可结合性也能更进一步地促进其二次开发，所谓技术的革新与共享带来科技的进步。

4.2 可扩展之处

该光立方有许多可扩展之处，例如可外接各种传感器，并通过图形显示；可设计上位机并实现远程控制或开发各种娱乐游戏；可植入人工智能算法来识别图像、音频……

此光立方的实现，给三维显示系统的设计提供了更多的技术方案选择。同时可在其他设计领域得到启发式式应用。比如，可应用于酒店大厅灯饰、广场灯饰，将每一条 LED 灯条置于大型的广场或者草地，每一条上 LED 灯的亮灭模拟自然界中的萤火虫，整个空间内的“萤火虫”飞舞，实现人与自然和谐相处的美好愿景。

4.3 心得体会

比赛过程中，时间紧任务重，我们同时还承担这繁重的学业课程。这一个月来，我们三个人风雨兼程，从电路原理图设计、PCB 打样、光立方制作以及软件代码的编写、仿真和调试，我们攻克了各种软件硬件上的问题，最终完成了系统，并且更提高了自己的专业水准和解决问题的能力。

这次竞赛同时也暴露出我们自身的一些问题，针对这些问题，做出以下总结：

1.心理素质是影响竞赛的关键因素。比赛不仅比技术，也比心理素质，技术　再高的人如果欠缺一定的心理素质，将会直接影响到自身的发挥。

2.团队精神至关重要。在作品准备的过程中，大家都会遇到不同的问题，经历不同的挑战，遇到问题要相互讨论，相互指出各自的不足，相互交流，团结作战，才能共同成长。

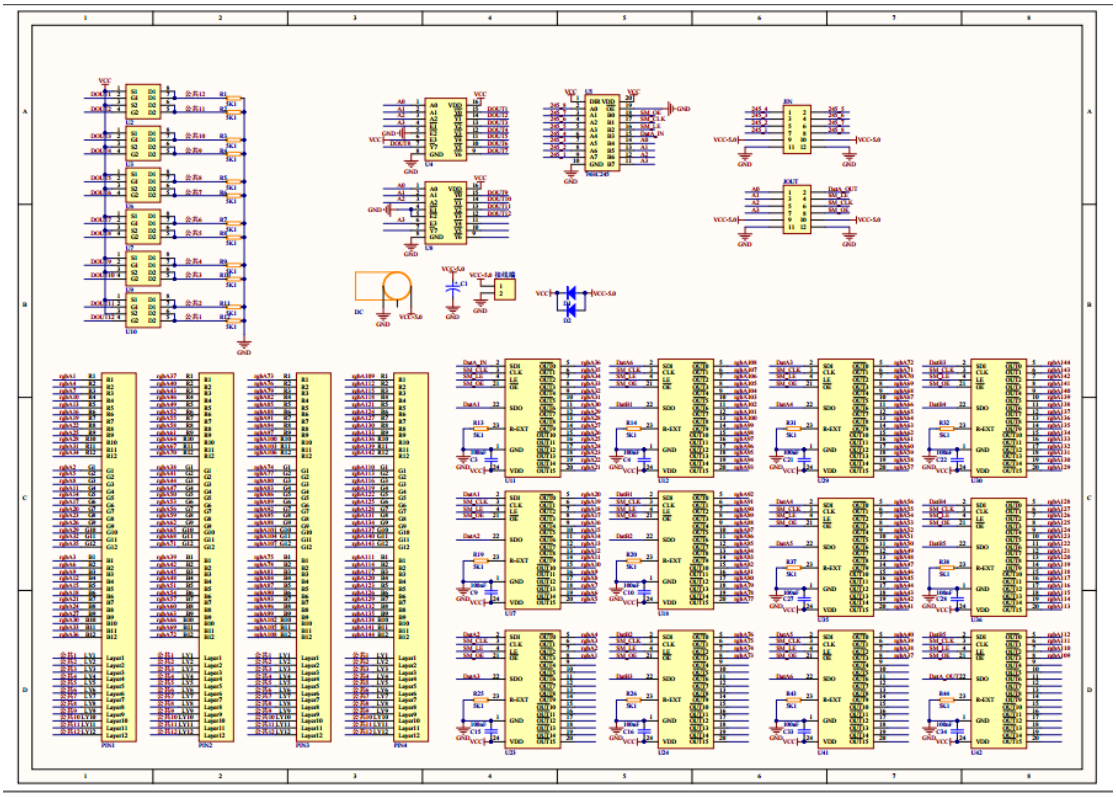
3.要学会自学，遇到难题和不懂得问题一定要自己去查资料，不要因为没有学过而灰心或失去信心。这次比赛我们最大的收获就是对 Verilog 语言及运用有了更深一步的了解，完善了以前的不足。深刻的认识到 HDL 语言的严谨性，失之毫厘谬以千里。

本次竞赛是一次理论和实践相结合的竞赛，充分体现了嵌入式开发人才的需求，给了我们一个自我提高和学习的好机会，为我们提供了一个展示自己的

大舞台。让我们懂得了竞赛不是一个人的战斗，而是靠整个团队的力量，以后的学习生活中，我们也会不断的积极进取，勇于拼搏，严于律己，宽以待人。

第五部分 附录/Appendix

附件 1：电路设计原理图



附图 1.1 控制底板原理图

附件 2：源程序代码

```
`timescale 1ns / 1ps
module spi_master(
          //置位与输出相应寄存器数据
             RESET,
            out_data,
         //系统时钟 clk，分频输出 clk_25m
            clk,
              clk_25m,
           //spi 接口
            mosi,
            miso,
            sclk,
        //LD3320 接口
             ld3320_cs,
             ld3320_irq,
```

```
            ld3320_reset,
            ld3320_wr
    );
wire ALL_START;
/*************************************/
        reg [1:0] addr;
        wire      wr;
        reg       rd;
/*************************************/

output ld3320_reset;
reg ld3320_reset;
output ld3320_wr;
reg ld3320_wr;
output ld3320_cs;
reg ld3320_cs;
inout ld3320_irq;

reg [7:0] in_data;
output [7:0]out_data=8'bx;
reg [7:0]out_data=8'bx;
/*************************************/
input RESET;
output clk_25m;
input clk;
inout miso;
inout mosi;
inout sclk;
 /*************************************/
reg sclk_buffer = 1;
//reg sclk_buffer = 0;
reg mosi_buffer = 0;
reg busy = 0;

reg [7:0]in_buffer = 0;
reg [7:0]out_buffer = 8'b10101010;
reg [7:0]clkdiv = 0;
reg [4:0]count = 0;

reg read_instructions=0;
reg write_instructions=0;
reg Read_Trigger;
reg Write_Trigger;
reg [1:0]Take_Data_Count;
```

```
/*************************************/
  clk_wiz_0 CLK25M
 (
  .clk_out1(clk_25m),     // output clk_out1
  .clk_in1(clk)           // input clk_in1
  );

vio_0 VIRTUAL_IO(
  .clk(clk),                 // input wire clk
  .probe_out0(ALL_START)  // output wire [0 : 0] probe_out0
);
//SPI receive_data
always@(rd or out_buffer or busy ) begin
    out_data = 8'bx;
    rd<=0;
end
 //SPI send_data
always@(posedge clk_25m) begin
    if(!busy) begin
        if(wr) begin
            case(addr)
                2'b00 : begin
                    in_buffer <= in_data;
                    busy <= 1'b1;
                    end
                2'b10 :   in_buffer <= clkdiv;
                default : in_buffer <= in_buffer;
            endcase
        end// end if(cs&&wr);
    end
    else begin
           if((count%2!=0)&&(count!=0)&&(count!=17)) begin
               mosi_buffer <= in_buffer[7];
               in_buffer <= in_buffer<<1;
           end

           if(count>0 && count<17) begin
               sclk_buffer <= ~sclk_buffer;
           end
           count <= count+1;
           if(count==17)begin
           Take_Data_Count<=Take_Data_Count+1;
           end
           if(Take_Data_Count==3)begin
```

```
                rd<=1;
                out_data <= out_buffer;
                Take_Data_Count<=0;
             end
            if(count >17) begin
                busy <= 0;
                count<= 0;
            end

     end
end

always@(negedge sclk_buffer) begin
    out_buffer = out_buffer<<1;
    out_buffer[0] = miso;
end
    assign wr=(write_instructions||read_instructions)&&!busy;
    assign sclk=sclk_buffer;
    assign mosi=mosi_buffer;
/**********************END SPI_MASTER*****************************/

/*****************************************************************/
/**************************START LD3320*************************/
/*****************************************************************/
//LD3320 write_reg read_reg
/*****************************************************************/
//读写寄存器 fixed order
parameter   write_order=8'b0000_0100;
parameter   read_order =8'b0000_0101;
parameter   take_data  =8'b0000_0000;
//状态机参数
parameter   IDLE=8'b0000_0000;
parameter   IDLE_NEXT=8'b0000_0001;

parameter   write_data1=8'b0000_0010;
parameter   write_data1_next=8'b0000_0011;
parameter   write_data2=8'b0000_0100;
parameter   write_data2_next=8'b0000_0101;

parameter   write_reg_addr=8'b0000_0010;
parameter   write_reg_addr_next=8'b0000_0011;
parameter   Take_reg_data =8'b0000_0100;
parameter   Take_reg_data_next =8'b0000_0101;
```

```
parameter   RETURN_START=8'b0000_0110;
parameter   RETURN=8'b0000_0111;
reg [7:0] Read_State;
reg [7:0] Write_State;
//读写寄存器数据
reg [7:0] data1;
reg [7:0] data2;
reg [7:0] reg_addr;
reg Write_reg_ok=1;
reg Read_reg_ok=1;
reg Write_Trigger_ok=0;
reg Read_Trigger_ok=0;
wire Write_reg_start;
wire Read_reg_start;
//中断
reg [20:0]Delay_count;
reg ProcessInt_ok=0;
reg [20:0]ProcessInt_count;
//读写寄存器
always@(posedge clk_25m) begin
//     if(RESET)begin
//     Write_State<=IDLE;
//     Read_reg_ok<=1;
//     Write_reg_ok<=1;
//     end
//     else
     if(busy)begin
     read_instructions<=0;
     write_instructions<=0;
     end
     else begin
     if(Write_Trigger)
     case(Write_State)
         IDLE:begin
         Write_reg_ok<=0;
         ld3320_cs<=0;
         ld3320_wr<=0;
         in_data<=write_order;
         write_instructions<=1;
         Write_State<=IDLE_NEXT;
         end
         IDLE_NEXT:begin
         Write_State<=write_data1;
         end
```

```
        write_data1:begin
        in_data<=data1;
        write_instructions<=1;
        Write_State<=write_data1_next;
        end
        write_data1_next:begin
        Write_State<=write_data2;
        end

        write_data2:begin
        in_data<=data2;
        write_instructions<=1;
        Write_State<=write_data2_next;
        end
        write_data2_next:begin
        Write_State<=RETURN_START;
        end
        RETURN_START:begin
        Write_Trigger_ok<=1;
        Write_State<=RETURN;
        end
        RETURN:begin
        ld3320_cs<=1;
        ld3320_wr<=1;
        Write_State<=IDLE;
        Read_reg_ok<=1;
        Write_reg_ok<=1;
        Write_Trigger_ok=0;
        end
    endcase
    if(Read_Trigger)
    case(Read_State)
        IDLE:begin
        Read_reg_ok<=0;
        ld3320_cs<=0;
        ld3320_wr<=0;
        in_data<=read_order;
        read_instructions<=1;
        Read_State<=IDLE_NEXT;
        end
        IDLE_NEXT:begin
        Read_State<=write_reg_addr;
        end
```

```
          write_reg_addr:begin
          in_data<=reg_addr;
          read_instructions<=1;
          Read_State<=write_reg_addr_next;
          end
          write_reg_addr_next:begin
          Read_State<=Take_reg_data;
          end
          Take_reg_data:begin
          in_data<=take_data;
          read_instructions<=1;
          Read_State<=Take_reg_data_next;
          end
          Take_reg_data_next:begin
          Read_State<=RETURN_START;
          end
          RETURN_START:begin
          Read_Trigger_ok<=1;
          Read_State<=RETURN;
          end
          RETURN:begin
          ld3320_cs<=1;
          ld3320_wr<=1;//在实际运用中 并未拉高
          Read_reg_ok<=1;
          Write_reg_ok<=1;
          Read_Trigger_ok<=0;
          Read_State<=IDLE;
          end
       endcase
   end
end
reg irq_trigger;
always@(posedge clk_25m or negedge ld3320_irq)begin
    if(!ld3320_irq)begin
    irq_trigger<=1;
    end
    else begin
    irq_trigger<=0;
    end
end
//触发与装载数据
reg [99:0] All_Init_Count=0;
reg ALL_Init_OK=0;
wire Read_reg_busy;
```

```
wire Write_reg_busy;
reg [7:0]The_Result;
reg Start_Get_Result;
reg [20:0]Delay;
assign Write_reg_busy=!Write_reg_ok;
assign Read_reg_busy=!Read_reg_ok;
always@(posedge clk_25m)begin
   if(Read_Trigger_ok)begin
    Read_Trigger<=0;end
   if(Write_Trigger_ok)begin
    Write_Trigger<=0;end
  if(ALL_START)begin
     if(!Write_reg_busy&!Read_reg_busy&!ALL_Init_OK)
        begin
            case(All_Init_Count)
//LD3320 LD_Init_Common
              0:begin

Read_Trigger<=1;reg_addr<=8'h06;All_Init_Count<=All_Init_Count+1;
              end
              1:begin
              All_Init_Count<=All_Init_Count+1;
              end
              2:begin

Write_Trigger<=1;data1<=8'h17;data2<=8'h35;All_Init_Count<=All_Init_
Count+1;
              end
              3:begin
              All_Init_Count<=All_Init_Count+1;
              end

              4:begin
              if(Delay==350)begin
               Delay<=0;All_Init_Count<=All_Init_Count+1;
              end
              else begin
               Delay<= Delay+1;
               All_Init_Count<=All_Init_Count;
              end
              end
              5:begin

Read_Trigger<=1;reg_addr<=8'h06;All_Init_Count<=All_Init_Count+1;
```

```
                end
                6:begin
                All_Init_Count<=All_Init_Count+1;
                end

                7:begin

Write_Trigger<=1;data1<=8'h89;data2<=8'h03;All_Init_Count<=All_Init_
Count+1;
                end
                8:begin
                All_Init_Count<=All_Init_Count+1;
                end
                9:begin
                if(Delay==350)begin
                 Delay<=0;All_Init_Count<=All_Init_Count+1;
                end
                else begin
                 Delay<= Delay+1;
                 All_Init_Count<=All_Init_Count;
                end
                end
                10:begin

Write_Trigger<=1;data1<=8'hcf;data2<=8'h43;All_Init_Count<=All_Init_
Count+1;
                end
                11:begin
                All_Init_Count<=All_Init_Count+1;
                end
                12:begin
                if(Delay==350)begin
                 Delay<=0;All_Init_Count<=All_Init_Count+1;
                end
                else begin
                 Delay<= Delay+1;
                 All_Init_Count<=All_Init_Count;
                end
                end
                13:begin

Write_Trigger<=1;data1<=8'hcb;data2<=8'h02;All_Init_Count<=All_Init_
Count+1;
                end
```

```
                14:begin
                All_Init_Count<=All_Init_Count+1;
                end
                15:begin
                Write_Trigger<=1;data1<=8'h11;
data2<=8'h09;All_Init_Count<=All_Init_Count+1;
                end
                16:begin
                All_Init_Count<=All_Init_Count+1;
                end
                17:begin
                Write_Trigger<=1;data1<=8'h1e;
data2<=8'h00;All_Init_Count<=All_Init_Count+1;
                end
                18:begin
                All_Init_Count<=All_Init_Count+1;
                end

                19:begin
                Write_Trigger<=1;data1<=8'h19;
data2<=8'h3f;All_Init_Count<=All_Init_Count+1;
                end
                20:begin
                All_Init_Count<=All_Init_Count+1;
                end
                21:begin

Write_Trigger<=1;data1<=8'h1b;data2<=8'h48;All_Init_Count<=All_Init_
Count+1;
                end
                22:begin
                All_Init_Count<=All_Init_Count+1;
                end

                23:begin

Write_Trigger<=1;data1<=8'h1d;data2<=8'h1f;All_Init_Count<=All_Init_
Count+1;
                end
                24:begin
                All_Init_Count<=All_Init_Count+1;
                end
                25:begin
                if(Delay==350)begin
```

```
                Delay<=0;All_Init_Count<=All_Init_Count+1;
               end
               else begin
                Delay<= Delay+1;
                All_Init_Count<=All_Init_Count;
               end
               end
               26:begin
               Write_Trigger<=1;data1<=8'hcd;
data2<=8'h04;All_Init_Count<=All_Init_Count+1;
               end
               27:begin
               All_Init_Count<=All_Init_Count+1;
               end
               28:begin

Write_Trigger<=1;data1<=8'h17;data2<=8'h4c;All_Init_Count<=All_Init_
Count+1;
               end
               29:begin
               All_Init_Count<=All_Init_Count+1;
               end
               30:begin

Write_Trigger<=1;data1<=8'hb9;data2<=8'h00;All_Init_Count<=All_Init_
Count+1;
               end
               31:begin
               All_Init_Count<=All_Init_Count+1;
               end
               32:begin

Write_Trigger<=1;data1<=8'hcf;data2<=8'h4f;All_Init_Count<=All_Init_
Count+1;
               end
               33:begin
               All_Init_Count<=All_Init_Count+1;
               end
               34:begin

Write_Trigger<=1;data1<=8'h6f;data2<=8'hff;All_Init_Count<=All_Init_
Count+1;
               end
               35:begin
```

```
                All_Init_Count<=All_Init_Count+1;
                end
////LD3320 LD_Init_ASR
                36:begin

Write_Trigger<=1;data1<=8'hbd;data2<=8'h00;All_Init_Count<=All_Init_
Count+1;
                end
                37:begin
                All_Init_Count<=All_Init_Count+1;
                end
//ProcessInt
    // if(!ld3320_irq&ALL_Init_OK&!ProcessInt_ok)begin
     if(irq_trigger&ALL_Init_OK&!ProcessInt_ok)begin
            if(!Write_reg_busy&!Read_reg_busy)begin
                case(ProcessInt_count)
                   0:begin

Read_Trigger<=1;reg_addr<=8'h2b;ProcessInt_count<=ProcessInt_count+1
;
                   end
                   1:begin
                     ProcessInt_count<=ProcessInt_count+1;
                   end
                   2:begin

Write_Trigger<=1;data1<=8'h29;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
                   end
                   3:begin
                     ProcessInt_count<=ProcessInt_count+1;
                   end
                   4:begin

Write_Trigger<=1;data1<=8'h02;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
                   end
                   5:begin
                     ProcessInt_count<=ProcessInt_count+1;
                   end
                   6:begin

Write_Trigger<=1;data1<=8'h2b;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
```

```
                    end
                    7:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    8:begin

Write_Trigger<=1;data1<=8'h1c;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
                    end
                    9:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    10:begin

Write_Trigger<=1;data1<=8'h29;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
                    end
                    11:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    12:begin
                      Write_Trigger<=1;data1<=8'h02;
data2<=8'h00;ProcessInt_count<=ProcessInt_count+1;
                    end
                    13:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    14:begin
                      Write_Trigger<=1;data1<=8'h2b;
data2<=8'h00;ProcessInt_count<=ProcessInt_count+1;
                    end
                    15:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    16:begin
                      Write_Trigger<=1;data1<=8'hba;
data2<=8'h00;ProcessInt_count<=ProcessInt_count+1;
                    end
                    17:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    18:begin
                      Write_Trigger<=1;data1<=8'hbc;
data2<=8'h00;ProcessInt_count<=ProcessInt_count+1;
```

```
                    end
                    19:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    20:begin

Write_Trigger<=1;data1<=8'h08;data2<=8'h01;ProcessInt_count<=Process
Int_count+1;
                    end
                    21:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    22:begin

Write_Trigger<=1;data1<=8'h08;data2<=8'h00;ProcessInt_count<=Process
Int_count+1;
                    end
                    23:begin
                      ProcessInt_count<=ProcessInt_count+1;
                    end
                    24:begin
                     if(Delay_count==50)begin
                     Delay_count<=0;
                     ProcessInt_count<=ProcessInt_count+1;
                     end
                     else begin
                     ProcessInt_count<=ProcessInt_count;
                     Delay_count<=Delay_count+1;
                     end
                    end
                    25:begin

Read_Trigger<=1;reg_addr<=8'hc5;ProcessInt_count<=ProcessInt_count+1
;
                    end
                    26:begin
                     if(Delay_count==1000)begin
                     Delay_count<=0;
                     ProcessInt_count<=ProcessInt_count+1;
                     end
                     else begin
                     ProcessInt_count<=ProcessInt_count;
                     Delay_count<=Delay_count+1;
                     end
```

```
                    end
                    27:begin

ProcessInt_count<=0;ProcessInt_ok<=1;Start_Get_Result<=1;
                    end
                endcase
            end
        end
    if(Start_Get_Result)begin
    The_Result<=out_data;
    Start_Get_Result<=0;
    ALL_Init_OK<=0;
    ProcessInt_ok<=0;
    end
end
//always @(The_Result)begin
//end
ila_0 SPI_ILA (
   .clk(clk),               // input wire clk
   .probe0(out_data),       // input wire [7:0]  probe0
   .probe1(mosi),           // input wire [0:0]  probe1
   .probe2(miso),          // input wire [0:0]  probe2
   .probe3(sclk),            // input wire [0:0]  probe3
   .probe4(Read_Trigger), // input wire [0:0]  probe4
   .probe5(Write_Trigger),// input wire [0:0]  probe5
    .probe6(ld3320_irq),       // input wire [0:0]  probe6
    .probe7(ALL_Init_OK)        // input wire [0:0]  probe7
);
endmodule
```