同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2251934

姓　　名 何锦洋

专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

本实验为基于 FPGA、MP3 和 OLED 制作的一个可视化音乐播放系统。
核心功能是音乐播放，通过 FPGA 内置的 RAM 存放三首歌的音乐格式文件，通过 mp3 进行播放，而 oled 实现简单的显示功能。

详细使用方法：

FPGA 的 RAM 中内置了 3 首歌曲。下板完成后，mp3 自动播放第一首歌曲，并且 oled 显示歌曲相关信息。使用操作如下：

1、通过按开发板的上/下按键，可以调整 mp3 音量。

2、按左/右键可以切换上一首/下一首歌曲。

3、按中间键可以暂停/播放音乐。

4、推动右下角复位开关可以对 mp3 和 oled 进行复位。

相关的显示信息如下：

1、播放歌曲时，开发板左下角的灯表示当前正在播放第几首歌曲，并且显示暂停图标，表示音乐正在播放，按下暂停键后可以暂停。

2、歌曲暂停时，开发板右下角信号灯亮起，说明正在暂停状态；oled 的歌曲序号滚动停止，方框中显示播放图标，表示音乐已暂停，按下暂停键后可以播放。

3、推动复位键后，开发板左下角指示灯回到第一首，oled 熄屏。

二、音乐播放可视化系统总框图

整个系统通过一个 Top 顶层控制，该层设计的接口
该层实现的功能

Top 层下设三个主要的子模块，依次是 MP3 模块，OLED 模块，音乐控制模块：

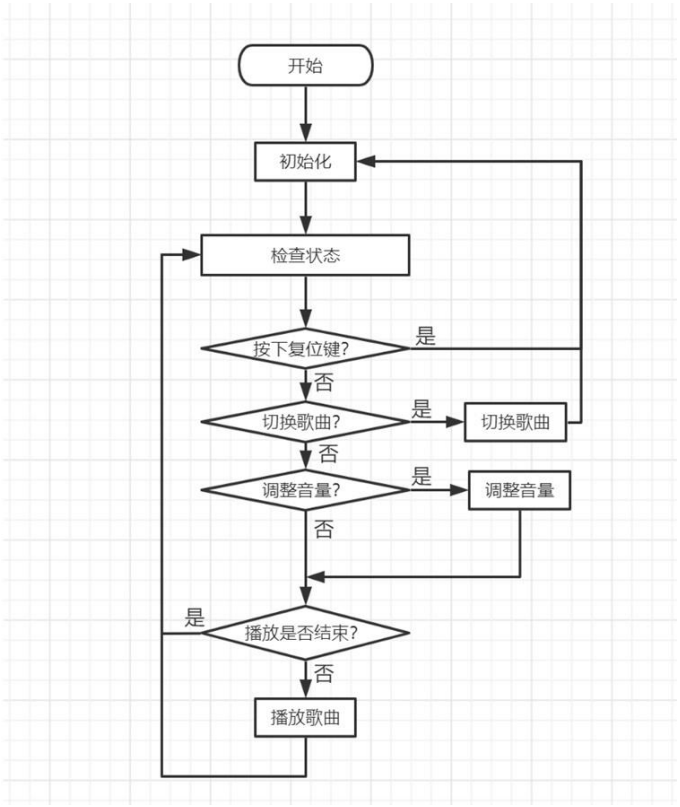
1、 MP3 模块：接受顶层控制器传来的歌曲内容、暂停信息、音量信息等，并播放歌曲。

2、 OLED 模块：接受顶层控制器传来的歌曲数、暂停信息等，显示相应图形。

3、 音乐控制模块：接受顶层传来的暂停，下一首，音量调节等信息，并对更改的信息进行处理

4、 三个模块的具体结构以及功能参考第三板块

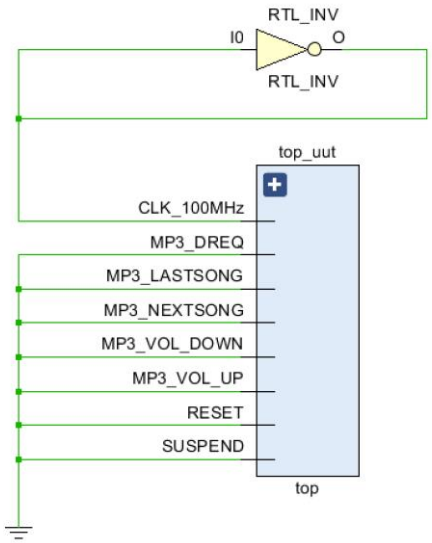
此外还包含分频模块，将 FPGA 板上的 100MHz 时钟信号按照各自需要的时钟频率分给不同的子模块。



三、系统控制器设计

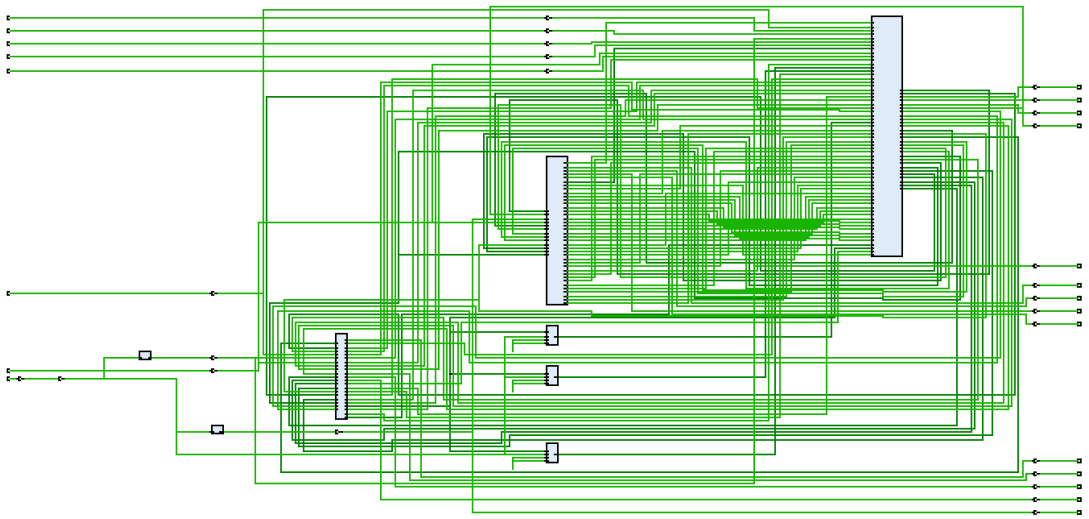
（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）

主流程图：



MUSIC 模块的状态转移表

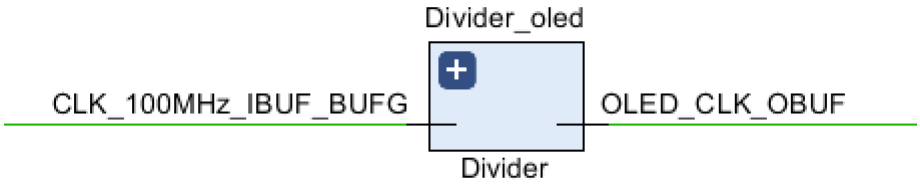
PS	转换条件	NS
hardReset（硬件重置）	无	initialisePre（初始化预备）
initialisePre（初始化预备）	无	initialise（初始化）
initialise（初始化）	无	checkState（检查当前状态）
checkState（检查当前状态）	RESET 有效	hardReset（硬件重置）
	改变歌曲有效，或者当前歌曲播放完毕	initialisePre（初始化预备）
	改变音量有效	adjustVolPre（调整音量预备）
	暂停有效	checkState（检查当前状态）
	以上信号都无效	displayMusic（播放音乐）
displayMusic（播放音乐）	无	checkState（检查当前状态）
adjustVolPre（调整音量预备）	无	adjustVol（调整音量）
adjustVol（调整音量）	无	checkState（检查当前状态）



四、子系统模块建模

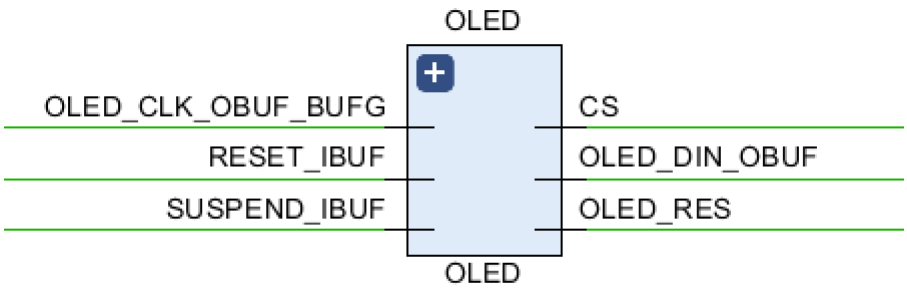
（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模，图文描述，描述清楚设计及实现的思路，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

1、 Divider 子模块：分频器。传入分频数和原始时钟，传出分频后的时钟。




```
module Divider(
    input I_CLK,    //输入时钟信号，上升沿有效     input rst,      //同步复
位信号，高电平有效     output reg O_CLK=0    //输出时钟
);     parameter defaultMultiple=2;   //默认分频倍数
     integer counter=0;     always@(posedge
I_CLK)     begin         if(rst==1)         begin             counter<=0;
            O_CLK<=0;         end         else         begin
if(counter==defaultMultiple/2-1)             begin                 counte
r<=0;                 O_CLK<=~O_CLK;             end             else
           counter<=counter+1;         end     end endmodule
```

2、 OLED 子模块



OLED 模块是辅助 MP3 模块进行一些基本的显示。

OLED 模块是 0.95 英寸，96RGB x 64 像素点的显示屏，显示原理为从内置的 Graphic Display Data RAM (GDDRAM)读取数据，并通过特定的成像原理，将像素点上的颜色映射到屏幕上。

OLED 的管脚说明：

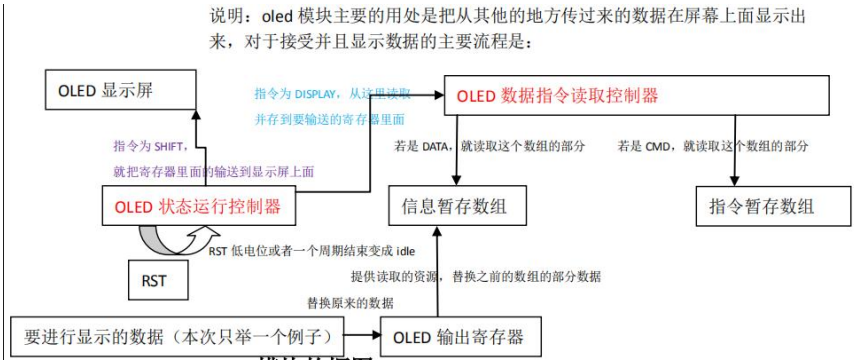
引脚号	标识	描述
1	VCC	电源正(3.3~5V 电源输入)
2	GND	电源地
3	NC	NC
4	DIN	模块数据输入
5	CLK	模块时钟信号输入
6	CS	模块片选信号，低电平有效
7	D/C	模块命令信号，低电平表示命令，高电平表示数据
8	RES	模块复位信号，低电平有效

根据不同管角的信号可以分为不同的状态：

Table 5 - Control pins of 6800 interface

Function	E	R/W#	CS#	D/C#
Write command	↓	L	L	L
Read status	↓	H	L	L
Write data	↓	L	L	H
Read data	↓	H	L	H

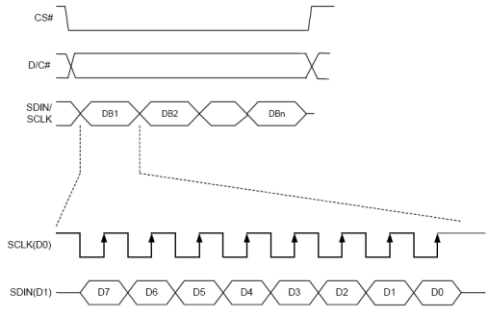
最为常用的状态是 CS 片选信号有效（低电平），模块命令信号 D/C 为低电平（传输命令），将指令通过 DIN 端口输入 OLED 的控制器。或者 D/C 信号为高电平，传入数据，数据存储在 OLED 的 GDDRAM 内置寄存器中，可以通过显示命令显示在屏幕上。



具体的通信遵循 SPI 传输协议，该串行接口由串行时钟 CLK, 串行数据 DIN, DC 和 CS 组成，DIN 信号在每个 CLK 的上升沿时刻，依次以高位在前低位在后地顺序被移至一个 8 位的移位寄存器， DC 在每第八个时钟周期被采样，并且移位寄存器里的数据被写入显示数据 RAM 或命令寄存器。

参考 SPI 接口文档

Figure 8 - Write procedure in SPI mode



由于 OLED 和 MP3 均遵循 SPI 协议，这里对 SPI 通信编程思路做一个介绍：

1.由于 OLED 模块在每个时钟的上升沿读取数据线,所以需要确保在每一个上升沿之前把数据线准备好,这里由于 FPGA 输入时钟周期远小于 OLED 模块允许的信号保持时间,除了分频之外,可以通过在每个低电平的中间(而不是每个下降沿)改变信号线,能够确保通信稳定.

2.由于 OLED 模块只要求在每个上升沿读取数据线,所以不同于 IIC 通信协议,这里对时钟线的空闲状态没有要求.如果时钟线空闲状态为高,则在第一个边沿(下降沿)到下一个边沿之间改变信号线;如果空闲状态为低,则需要在空闲状态就把数据准备好.

3.为了方便高层模块读取 SPI 通信的状态,设置一根 busy 线,当通信开始时置 busy 线为高,通信结束时置低,高层模块在需要发送数据时先读 busy 信号,若为低则直接发送,若为高则等到 busy 变为低再发送.

4.使用移位寄存器,每次发送高位，每发送一位,寄存器往左移一位

5.使用状态机完成通信:

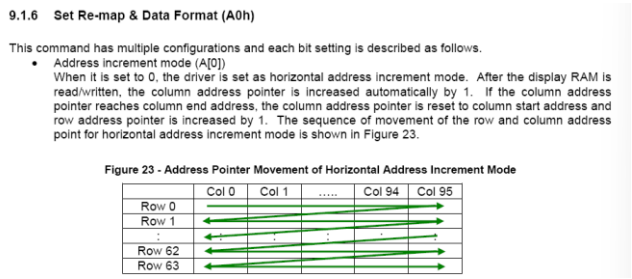
部分代码展示：

```
module SpiCtrl(
);
// 定义参数
    localparam Idle = 0;
    ……
// 定义变量
    reg [2:0]   state = Idle;       // 状态
    reg [7:0]   shift_register=0;   // 传输数据
    ……
// 初始化
// 输出函数
    assign SCLK = (counter < SCLK_DUTY) | CS;
// 状态转移
    always@(posedge clk)
        case (state)
        Idle: begin
            if (send_start == 1'b1)
                state <= Send;
        end
        ……
        endcase
// 计数器变化
// 内部函数
    always@(posedge clk)
        if (state == Idle) begin
            shift_counter <= 'b0;
            shift_register <= send_data;
            temp_sdo <= 1'b1;
        end
```

介绍完串口通信，接下来就是具体如何在屏幕上显示，我认为显示分为两种，一种是直接通过命令显示，另一种是通过数据传输显示。我将依次介绍：

然而在介绍显示前，有必要介绍下 OLED 的初始化部分。参考指令表，OLED 的初始化（重置）主要有以下几步：

指令 A0H，设置屏幕显示形式



9.1.6 Set Re-map & Data Format (A0h)

This command has multiple configurations and each bit setting is described as follows.

- Address increment mode (A[0])
  When it is set to 0, the driver is set as horizontal address increment mode. After the display RAM is read/written, the column address pointer is increased automatically by 1. If the column address pointer reaches column end address, the column address pointer is reset to column start address and row address pointer is increased by 1. The sequence of movement of the row and column address point for horizontal address increment mode is shown in Figure 23.

Figure 23 - Address Pointer Movement of Horizontal Address Increment Mode

	Col 0	Col 1	.....	Col 94	Col 95
Row 0					
Row 1					
:	:	:	:	:	:
Row 62					
Row 63					

本次使用的显示方式是纵坐标按从左到右从小到大排列，但是横坐标从下到上从小到大

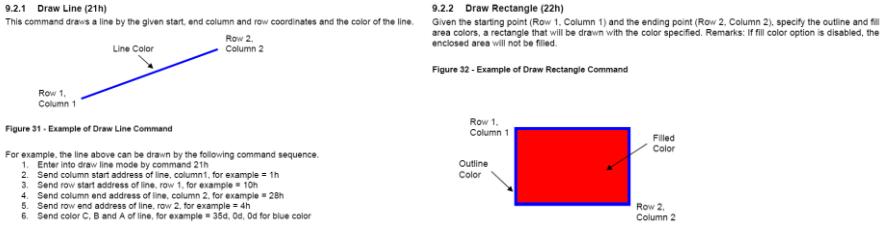
排列。其余的行偏移和列偏移还有颜色编码格式等均保留默认值。

除了 Remap 命令还有其他的初始化命令如设置显示起始行数，偏移量，显示模式，显示帧率等。为了方便起见基本都保留均值。最终呈现出如下的指令码：

```
ae_a0_74_a1_00_a2_00_a4_a8_3f_ad_8e_b0_0b_b1_31_b3_f0_8a_64_8b_78_8c_64
_bb_3a_be_3e_87_06_81_91_82_50_83_7d_af
```

初始化结束后，通过 SPI 协议传输数据：

首先是通过命令控制，命令控制的绘图显示主要指令有两个，一个是绘制直线，一个是绘制图形。

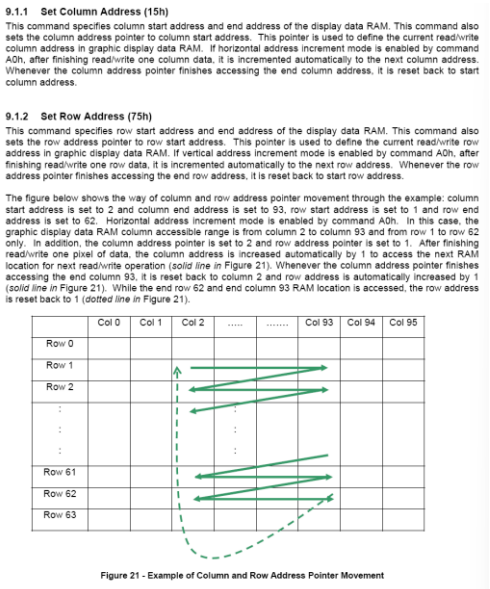


两者指令类似，均是通过给出具体指令以及相应的行列位置，像素点颜色即可完成绘制。

指令控制还有很多其他直接对屏幕操作的，如清屏，拷贝，轮转等，在后面会着重介绍轮转。

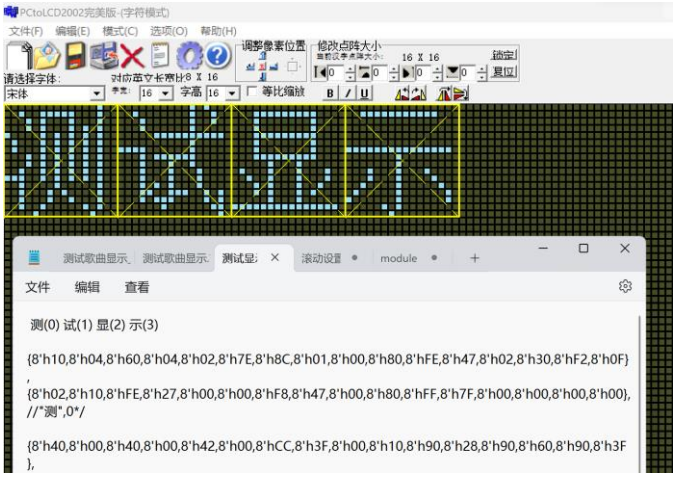
第二种绘制方式是直接对屏幕内置寄存器传输数据，传输数据还是命令的不同在于 D/C 线是高电平还是低电平。然而在传输数据前还需要通过命令对起始行列和终止行列以及数据传入的方式（纵向或者横向）进行设置。

文档中首先给出了需要传输数据的基本指令：



通过 15H 以及 75H 设置起始和终止的行列数，默认按照先列后行的方式存储，当遍历到终点时会自动退回起点，避免越界。

在设置完遍历方式之后就是通过 SPI 协议将需要传入的数据（像素数据）一个一个通过 OLED_DIN 信号线传入。由于传入的是像素点（甚至可以说是像素点的某一个颜色的一个分量（8 位），而在 FPGA 看来，传入的则是一个个 01 信号），所以我们需要借助专业软件将我们需要的（宏观上的）图片或者文字转成微观上的二进制代码。这里利用的是 PCToLCD2002 软件：



顺带一提：LCD 是另一种显示方式，LCD 屏幕和 OLED 屏幕主要区别是在硬件上对像素点显示的不同，而在软件层面，二者都是通过对像素点进行赋值从而显示出对应的图像。

通过专业软件生成的二进制值我们可以定义常变量参数，从而被 vivado 综合，传输到 OLED 显示屏上并显示。

由于 OLED 需要大量的指令，额外包含一个头文件，文件内存放指令对应的常量数组。

代码展示：

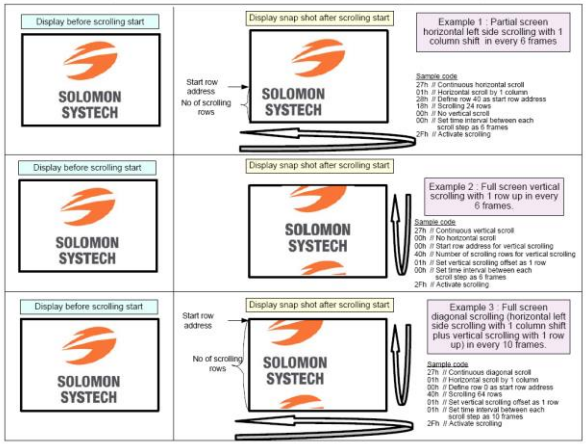
```
//滚动设置
    parameter scrollActiveCommand=8'h2f;        //滚动有效
    parameter scrollDeactiveCommand=8'h2e;      //滚动无效
    parameter scrollSettingCommand={  //长度 48,滚动的设置
        8'h27,  //设置滚动模式
        8'h01,  //水平滚动
        8'd39,  //滚动的起始行
        8'd24,  //滚动持续的行数
        8'h00,  //无垂直滚动
        8'h00   //速度
    };
    //颜色设置
    parameter songColor=24'hC7EDCC;//ec_af_13;  //song 的颜色
    parameter songNumColor=24'hDCE2F1;   //歌曲标号的颜色
    parameter lastNextColor=24'hFAF9DE;//e1_f5_07;  //上一首和下一首的标志的颜色
    //清除指令
```

经过以上的步骤基本能实现简单的绘图，清空屏幕等过程（不过前提是得确保通信没有问题），一旦通信出现时序错位或者数据缺失，将会造成很大的后果。

接下来我将接受一种比较高级的显示方式——滚动显示。

滚动显示在软件层面实现起来较为困难，可能需要将整块的数据内容做整体迁移而且还可能出现数据丢失。然而 OLED 本身提供了一种滚动的方案：

Figure 35 - Examples of Continuous Horizontal and Vertical Scrolling command setup



可以看到，指令中并不涉及对列的设置。所以，oled 的滚动，如果想限制范围，仅可以行为单位进行限制，而不可以限制列。也就是说，如果想让某几列保持滚动，某几列静止，那么 oled 的滚动命令是无法实现此功能的。

其次，指令 2E 和 2F 分别表示禁用滚动和启用滚动。这里也涉及到一个非常重要的问题：若想要修改屏幕的当前内容，则必须在滚动禁用的状态下进行；当滚动启用时，不允许修改屏幕内容。否则，屏幕将会出现步数较大的垂直滚动现象，造成错误。

还有一点需要注意的是，每次启用滚动时，最好都发送一遍滚动设置指令。否则，在多次的启用—>禁用—>启用中，屏幕滚动效果会出现混乱。

滚动的具体代码

```
//滚动设置
parameter scrollActiveCommand=8'h2f;        //滚动有效
parameter scrollDeactiveCommand=8'h2e;      //滚动无效
parameter scrollSettingCommand={  //长度 48,滚动的设置
    8'h27,  //设置滚动模式
    8'h01,  //水平滚动
    8'd39,  //滚动的起始行
    8'd24,  //滚动持续的行数
    8'h00,  //无垂直滚动
    8'h00   //速度
};
```

在确定了 OLED 基本的功能之后，接下来就是确定状态。

状态转移表如下：

PS	转换条件	**NS**
reset（重置）	无	初始化预备
initialisePre（初始化预备）	无	初始化
initialise（初始化）	无	清屏
clearScreen（清屏）	RESET 无效	发送绘图指令
checkState（检查当前状态）	RESET 有效	重置

	改变歌曲有效	改变歌曲名预备
	暂停有效	改变播放状态预备
	以上信号都无效	检查当前状态
changeSongNamePre（改变歌曲名预备）	无	绘制单词 song
drawSong（绘制单词 song）	无	发送绘图指令
changeDisplayModePre（改变播放状态预备）	无	发送绘图指令
sendCommand（发送绘图指令）	无	检查当前状态

```
//小屏状态机设置
localparam  reset                =9'b100_000_000,//重置
            initialisePre        =9'b010_000_000,//初始化预备
            initialise           =9'b001_000_000,//初始化
            clearScreen          =9'b000_100_000,//清空窗口
            drawSong             =9'b000_010_000,//画 song 这几个字母
            checkState           =9'b000_001_000,//检查状态
            changeDisplayModePre=9'b000_000_100,//画暂停/播放状态准备
            changeSongNamePre    =9'b000_000_010,//画歌曲名的准备阶段
            sendCommand          =9'b000_000_001;//发送绘图指令
reg [8:0]state=reset;//初始化为重置状态
```

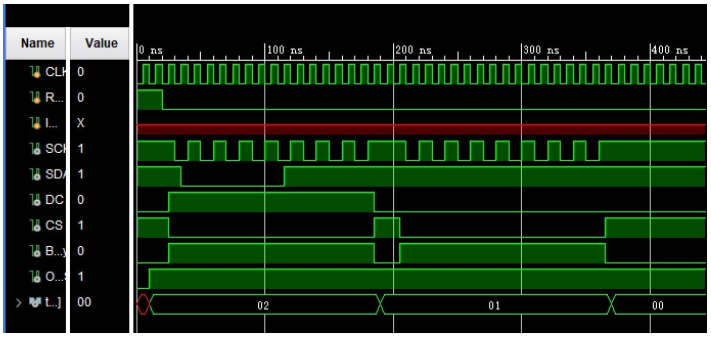
3、 MP3 子模块：本模块是 mp3 播放器实现的核心模块，几乎所有 MP3 的功能都在本模块中实现

MP3 模块实现的关键技术是 spi 通信方法。VS1003B MP3 Board 模块通过 SPI 接口与外部控制器连接， VS1003 的控制以及音频数据都是通过 SPI 接口进行。下表反应了 VS1003B 的 spi 信号线功能：

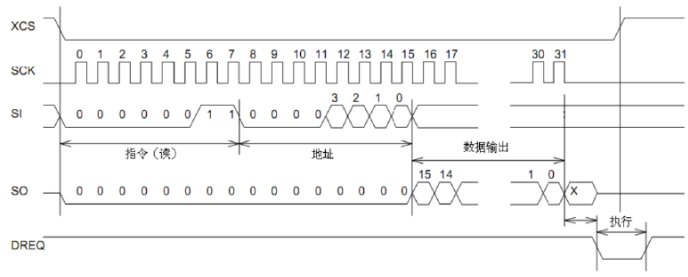
SDI 管脚	SCI 管脚	描述
XDCS	XCS	低电平有效片选输入，高电平强制使串行接口进入 standby 模式，结束当前操作。SO 变成高阻态。如果 SM_SDISHARE 位设置为 1，则不使用 XDCS，而有 XCS 内部反相后代替 XDCS。不过不推荐这种设置方式。
SCK		串行时钟输入。 SCK 在传输的时候可以被打断，但是必须保持 XCS/XDCS 低电平不变，否则传输将中断。
SI		串行数据输入。在片选有效的情况下，SI 在 SCK 的上升沿处采样。所以， MCU 必须在 SCK 的下降沿上更新数据
SO		串行输出， 在读操作时， 数据在 SCK 的下降沿处从此脚移出；在写操作时为高阻态。

MP3 的数据传输分为传入数据和传入命令两种类型，分别对应 sdi 和 sci。sdi 是标准的 spi 通信，在 DREQ 信号有效时，主机有序地向 MP3 发送音频文件，MP3 自动完成解码播放功能。sci 串行总线命令接口包含了一个指令字节、一个地址字节和一个 16 位的数据字。读写操作可以读写单个寄存器。

由于 MP3 的通信协议和 OLED 基本保持一致，所以此处不过多赘述，展示两者 SCI 协议图片以及 testBench 仿真结果。



SCI 读时序如图所示：




```
module mp3 (     input RESET,     input MP3_SCLK,
//mp3 的工作时钟          input MP3_DREQ,
     input [31:0]SongNow,    //当前的歌曲     input [31:0]Memory,     //当
前歌曲的内容      input [31:0]MusicSize,  //当前歌曲大小      input
IS_SUSPENDING,    //是否暂停     input [7:0]VOLUME,      //音量大小
     output reg [31:0]MemAddr,       //歌曲地址     output reg
MP3_MOSI,     output reg MP3_xCS,     output reg MP3_xDCS,     output reg
MP3_xRSET
```

```
);

 状态机：
    //状态机，以及其相应设置
    localparam  hardReset       =7'b1000000,    //硬件复
位                  initialisePre   =7'b0100000,    //初始化准备阶
段                  initialise      =7'b0010000,    //初始
化                  displayMusic    =7'b0001000,    //播放音
乐                  checkState      =7'b0000100,    //检查当前状
态                  adjustVolPre    =7'b0000010,    //调整音量准备阶
段                  adjustVol       =7'b0000001;    //调整音量
    reg [6:0]state=hardReset;       //标识当前状态，初始化为硬件复位

always 语句：时钟下降沿敏感（关键）
always@(negedge MP3_SCLK)   //数据在上升沿被采样，所以在下降沿写入

 checkState 状态：  checkState:       //检查当前状态
    begin
        //状态机转
换          if(RESET)             state<=hardReset;         else
if(lastSong!=SongNow)     //歌曲切
换              state<=initialisePre;       //直接进入初始化预备状态,即可
完成歌曲切换
        else if((volSetting[7:0]!=VOLUME)&&MP3_DREQ) //按下了音量键
            state<=adjustVolPre;        //进入调整音量预备状态
        else if(!IS_SUSPENDING&&MP3_DREQ)   //不处在暂停状态,并且可
以发送数据
            state<=displayMusic;        //进入播放状态          else    //
否则,该周期滞空
            ;
    end
```

4、 最后一个模块是控制模块，实现从 FPGA 的按键获取暂停，播放下一首和音量调节信号。该模块设计基本的时序逻辑和状态切换，此外设计防抖动功能，对每一个基本按键设置 CD 时间。

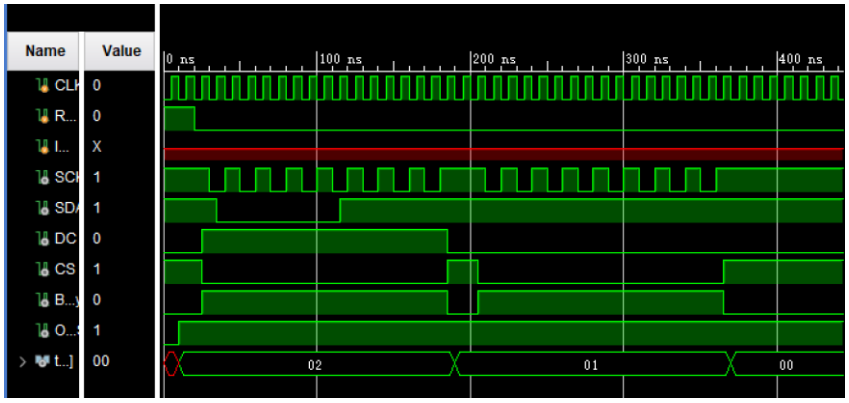
```
module MusicControl(
    input RESET,
    input MP3_SCLK,
    input SUSPEND,      //暂停按钮
    音量调整，    歌曲调整

    output reg IS_SUSPENDING=0,       //正处在暂停状态. 1 为暂停, 0 为播放
```

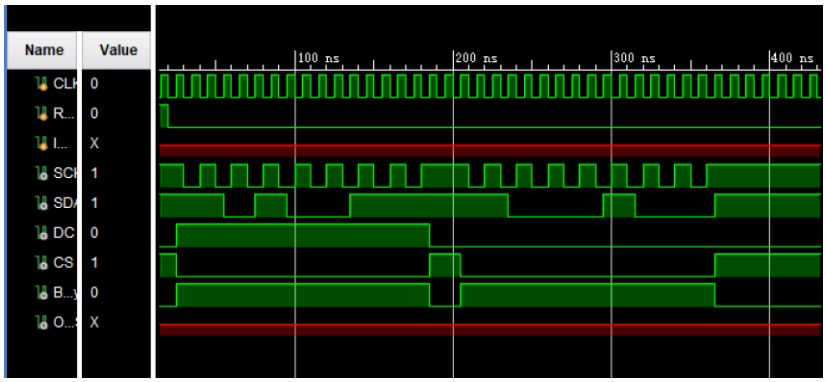
```
    output reg[7:0]VOLUME=0,     //输出当前音量
    output reg [31:0]SongNow=0  //初始时歌曲为第 0 首
);
    改变音量的冷却时间,此处为 0.5s
    localparam FixedVolCd=500000;
    ……

    always@(posedge MP3_SCLK) begin  //为避免与 mp3 模块发生冲突,此处在上升沿修改暂停
        if(RESET) 清零处理
        if(!RESET && SUSPEND) begin
            if(!isChange) 不可设置暂停
        end
        else isChange <= 0;

        if(!RESET && (MP3_VOL_UP || MP3_VOL_DOWN)) begin
            if(Volcd>=FixedVolCd)
            begin
                if(MP3_VOL_UP)
                    VOLUME<=(VOLUME==8'h0)?8'h0:VOLUME-FixedStep;
                else
                    VOLUME<=(VOLUME==8'hff)?8'hff:VOLUME+FixedStep;
                Volcd<=0;  //重新开始 cd
            end
            else Volcd<=Volcd+1;
        end
        else Volcd <= FixedVolCd;   //如果没有按下音量键,则下次改变有效
        ……
```

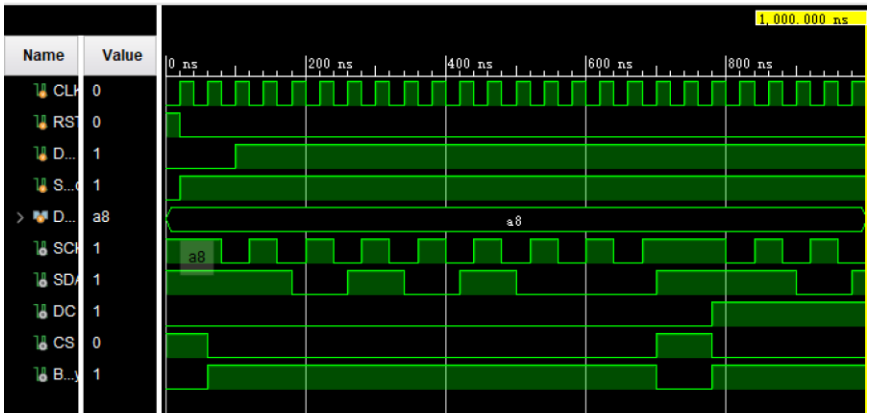
五、测试模块建模

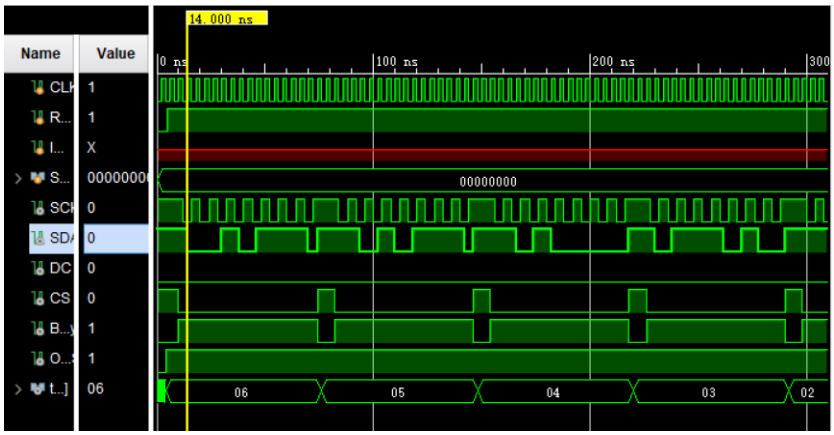
（要求列写各建模模块的 test bench 模块测试逻辑，不要在此处放 verilog 代码，所有的 verilog 代码在附录部分）

在完成自顶向下的设计之后便是自底端向上的实现过程，首先完成基本模块的测试过程，其中最为重要的模块是通信模块，以 OLED_SPI 通信模块为例，介绍通信模块的测试模块的建模过程，而对于其他模块如 MP3 音乐播放以及 OLED 屏幕娴熟，实现效果主要依靠下板后的显示，不能通过仿真波形图检验。









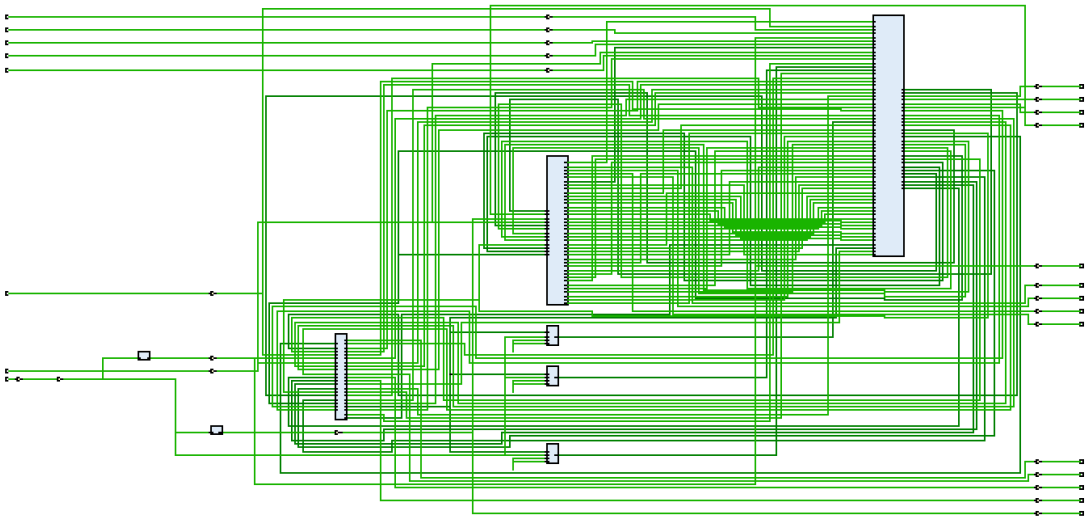
上面四幅仿真结果图依次实现的是在时钟信号下，单 OLED_SPI 调试，结合 OLED 控制的调试，其又包含直接赋值的调试，通过 cnt 以及缓存数组进行传输的调试，以及通过状态机切换的调试。

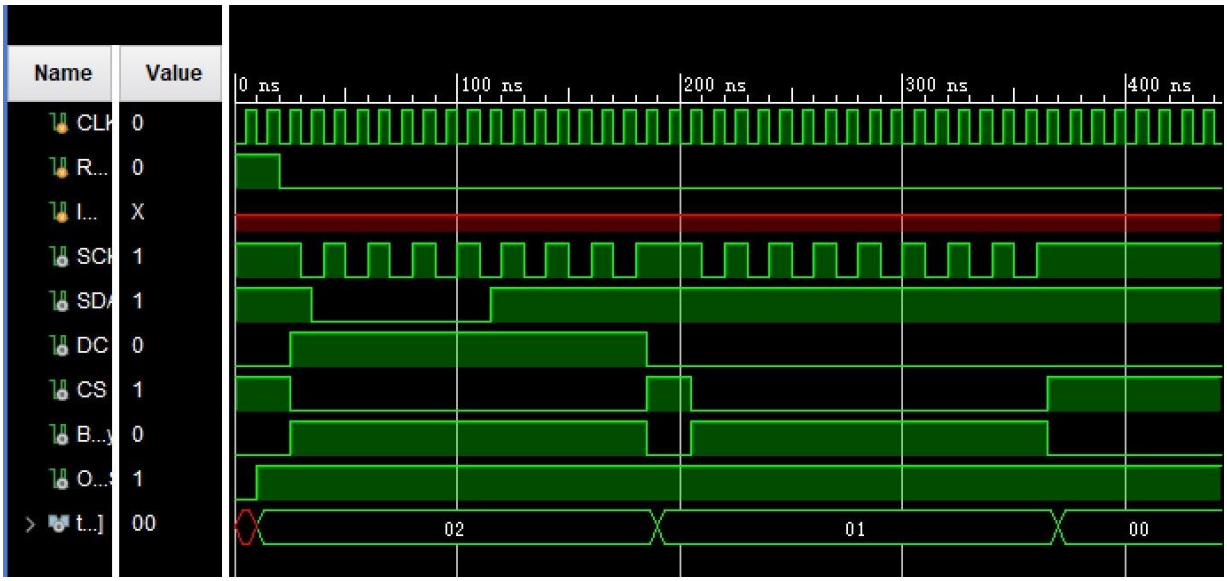
最终实现的效果是依次向 OLED_DIN 中传入 2F 和 AE，分别对应着初始化操作的清空滚动

状态以及设置屏幕初始化参数。

六、实验结果

（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

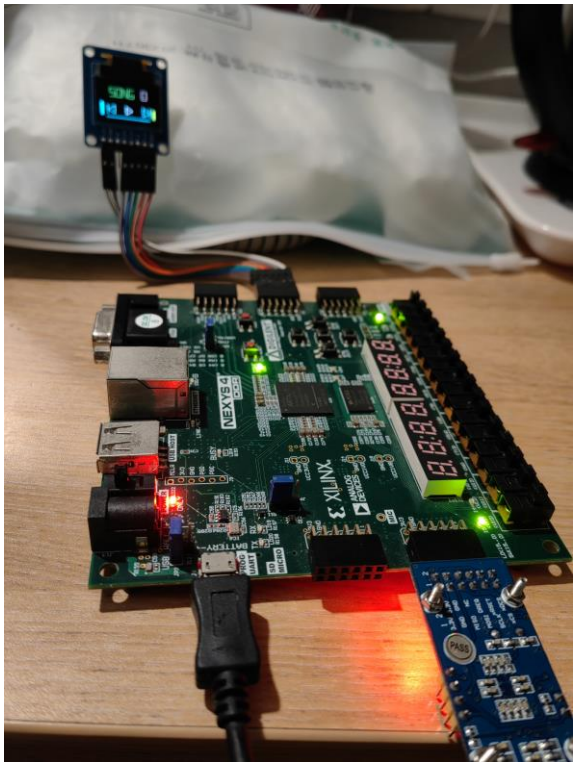


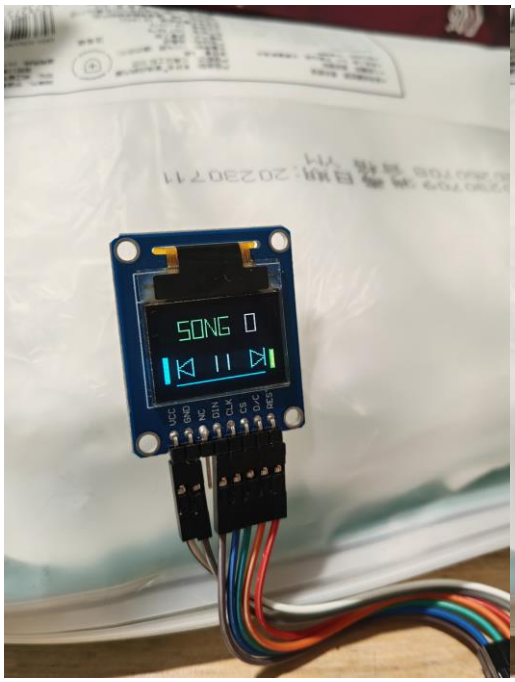


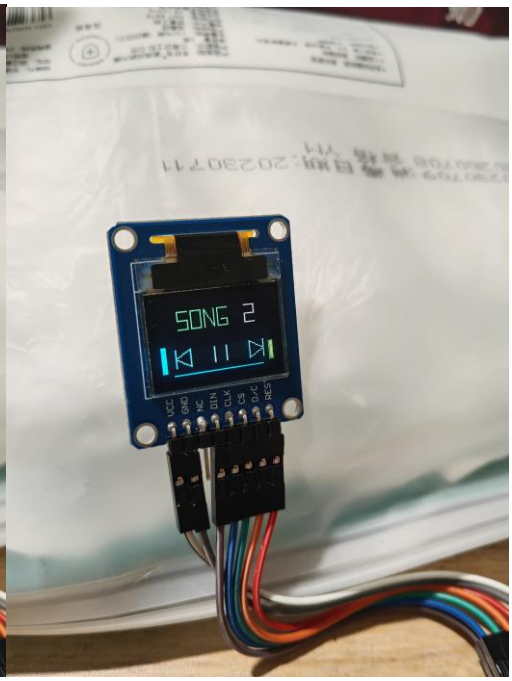
下板结果图：

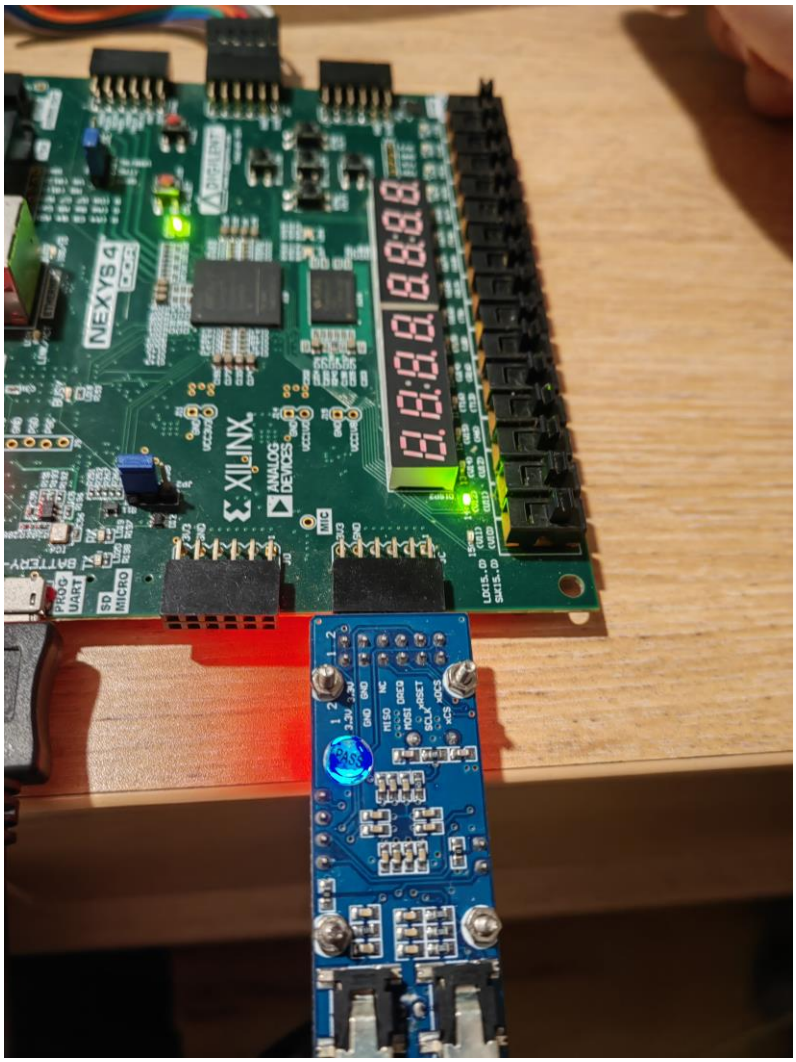
下板成功后，自动播放第 1 首歌曲（也就是 SONG 0），开发板左下角显示当前歌曲序号，暂停键按下后，歌曲暂停播放，右下角暂停信号灯亮起，并且显示按下播放。按下下一首键时，歌曲切换到下一首，左下角歌曲指示灯跳转到第二首（SONG 1），置右下角的重置信号为高时，MP3 和小屏进入重置状态，MP3 停止播放，小屏清屏。重置信号拨回低时，MP3 音量回到最大，歌曲序号回到

SONG 0。









七、结论

在开发过程中，时序问题显得尤为关键。这不仅仅包括了部件在何种时钟频率下能够正常运作，更涉及到每个步骤的先后次序、执行规则、数据写入时机以及控制信号变更的顺序等等复杂方面。然而，由于时间紧迫和对硬件开发经验的欠缺等原因，时序问题只能在实际实现过程中逐步解决。

相对于 VGA，0.95 寸 OLED 的开发难度更大。受限于资料匮乏，除了英文手册外几乎找不到其他参考资料，使得开发变得相当具有挑战性。由于 OLED 只有在时序和步骤等方面没有问题时才能在屏幕上显示反馈，这对于初学者来说增加了开发的难度。其次关于 SPI 通信协议的理解以及实现，也有很大难度。

在 MP3 的开发中，由于存储歌曲需要大量存储单元，仅仅依赖于有限的开发板存储空间是不够的。如果能够完成 SD 卡等存储设备的开发，将对 MP3 的进一步开发提供非常有利的条件。

开发过程遇到的常见问题：RESET 低电平有效和高电平有效冲突，线网型变量被多个控制。

很多问题是之前模块设计时遗漏的，因此需要每一个模块精确的设计，减少耦合度，能够独立测试，最后统一整合。同时做好版本管理，能及时回退到之前的版本上。

此外，在开发过程中，更加应该充分利用仿真波形进行调试。每次都进行下板调试不仅耗时较多，而且一旦出现错误，可能无从查找。因此，多使用仿真波形图进行调试，必要时可以采用逻辑分析仪等设备进行调试，将会更加高效和可靠。

八、心得体会及建议

1. 课程收益

数字逻辑是我首次接触硬件课程，这为我提供了对计算机的全新理解。在此之前，我只知道计算机底层由 0 和 1 组成，但无法将二进制与丰富的软件界面联系起来。数字逻辑课程帮助我建立了从门电路到寄存器、内存，再到软件等各个层次的基本硬件知识结构。

在着手大作业时，我陷入困境。通过学习数字系统开发方法和自顶向下设计，我理解到设计数字系统需要在开始实施之前对整个系统有充分的理解。此外，大作业的开发过程还使我深入了解了 Vivado 等软件的功能，并掌握了更多的测试台和调试方法。

本次大作业大大提高了我查阅英文资料和阅读能力。在开发 OLED 时，由于只有英文资料，我努力克服了阅读上的困难。尤其在实现 OLED 初始化功能时，由于缺乏参考资料，我通过不断修改代码和调试，摸索出了正确使用指令的方法。

总体而言，大作业的开发过程对我来说是一个充满挑战的新经验，锻炼了我的耐心、资料查阅和硬件调试能力。

2. 对课程的建议

大作业的难点在于起点高，对于没有经验的同学来说上手很难，尤其是阅读资料的方法。建议老师可以介绍外围部件的具体开发方法，并指导如何有效阅读资料。可以通过提供一些公共部件的开发代码和实验报告等，帮助学生更快上手开发。

同时，对于使用较多的公共部件，如果往届学生已经完成，可以提供一些开发代码和实验报告。例如，在 MP3 开发中，如果提供了开发好的 SD 卡代码，可能会帮助同学们取得更好的成果。

此外，建议在平时的课程小作业中多增加一些状态转换，数据传输，以及多模块之间的调用的作业，避免在上手大作业时难度过大。

九、附录

（该部分放 verilog 代码，包括所有设计文件和测试文件）

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////////
// Company:
// Engineer:
//
// Create Date: 2024/01/14 18:45:04
```

```
// Design Name:
// Module Name: MusicControl
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module MusicControl(
    input RESET,
    input MP3_SCLK,
    input SUSPEND,      //暂停按钮
    input MP3_VOL_UP,           //音量调整
    input MP3_VOL_DOWN,
    input MP3_LASTSONG,         //歌曲调整
    input MP3_NEXTSONG,

    output reg IS_SUSPENDING=0,       //正处在暂停状态. 1 为暂停, 0 为播放
    output reg[7:0]VOLUME=0,     //输出当前音量
    output reg [31:0]SongNow=0  //初始时歌曲为第 0 首
);

    localparam FixedVolCd=500000;     //改变音量的冷却时间,此处为 0.5s
    localparam FixedSongCd=1000000;     //改变歌曲的冷却时间,此处为 1s
    localparam FixedStep=8'h8;        //每次改变音量的步长,注意必须为 2 的次
幂
    localparam songNum=3;               //歌曲总数

    reg isChange=0;
    integer Volcd=FixedVolCd;
    integer Songcd=FixedSongCd;

    always@(posedge MP3_SCLK) begin  //为避免与 mp3 模块发生冲突,此处在上升沿
修改暂停
        if(RESET) begin
            isChange<=0;
```

```
            IS_SUSPENDING<=0;   //重置后,不暂停
            Volcd<=FixedVolCd;   //重置后改变有效
            VOLUME<=8'h0;         //重置后改变为最大音量
            Songcd<=FixedSongCd;   //重置后改变有效
            SongNow<=0;         //重置后改变为第 0 首
        end
        if(!RESET && SUSPEND) begin
            if(!isChange) begin
                IS_SUSPENDING<=~IS_SUSPENDING;
                isChange<=1;      //不可设置暂停
            end
        end
        else isChange <= 0;

        if(!RESET && (MP3_VOL_UP || MP3_VOL_DOWN)) begin
            if(Volcd>=FixedVolCd)
            begin
                if(MP3_VOL_UP)
                    VOLUME<=(VOLUME==8'h0)?8'h0:VOLUME-FixedStep;
                else
                    VOLUME<=(VOLUME==8'hff)?8'hff:VOLUME+FixedStep;
                Volcd<=0;  //重新开始 cd
            end
            else Volcd<=Volcd+1;
        end
        else Volcd <= FixedVolCd;   //如果没有按下音量键,则下次改变有效

        if(!RESET && (MP3_LASTSONG||MP3_NEXTSONG)) begin
            if(Songcd>=FixedSongCd) begin
                if(MP3_LASTSONG)
                    SongNow<=(SongNow==0)?0:SongNow-1;
                else
                    SongNow<=(SongNow==songNum-1)?0:SongNow+1;
                Songcd<=0;  //重新开始 cd
            end
            else Songcd<=Songcd+1;
        end
        else Songcd<=FixedSongCd;   //如果没有按下改变歌曲键,则下次改变有效
    end
endmodule


//滚动设置
    parameter scrollActiveCommand=8'h2e;       //滚动有效 //2f
```

```
    parameter scrollDeactiveCommand=8'h2e;      //滚动无效
    parameter scrollSettingCommand={  //长度 48,滚动的设置
        8'h27,  //设置滚动模式
        8'h01,  //水平滚动
        8'd39,  //滚动的起始行
        8'd24,  //滚动持续的行数
        8'h00,  //无垂直滚动
        8'h00   //速度
    };
    //颜色设置
    parameter songColor=24'hC7EDCC;//ec_af_13;  //song 的颜色
    parameter songNumColor=24'hDCE2F1;   //歌曲标号的颜色
    parameter lastNextColor=24'hFAF9DE;//e1_f5_07;  //上一首和下一首的标志
的颜色
    //清除指令
    parameter clearDisplayCommand={  //长度 40, 清除播放/暂停标识的指令
        //这里详细地展示了一个清除指令的构成
        8'h25,      //清除命令
        8'd43,      //起始列
        8'd10,      //起始行
        8'd55,      //结束列
        8'd24       //结束行
    };
    parameter clearLastSongCommand={ //长度 40, 清除上一个歌曲信息
        8'h25,      //清除命令
        8'd00,      //起始列
        8'd39,      //起始行
        8'd95,      //结束列
        8'd55       //结束行
    };
    parameter clearFullScreenCommand={ //长度 40, 清空屏幕
        //这里详细地展示了一个清除指令的构成
        8'h25,      //清除命令
        8'h00,      //起始列
        8'h00,      //起始行
        colMax,     //结束列
        rowMax      //结束行
    };
    //图形指令
    parameter drawBlockCommand={  //长度为 88, 画方格的设置
        8'h22,
        8'd35,
        8'd06,
        8'd63,
```

```
        8'd28,
        24'h00_ff_ff,   //边框颜色
        24'h00_00_00    //填充颜色
    };
    parameter  draw0Command={  //长度为 88，画三个数字的命令，最长 320
                    8'h22,
                    8'd72,
                    8'd40,
                    8'd80,
                    8'd54,
                    songNumColor,   //边框颜色
                    24'h00_00_00    //填充颜色
                },
                draw1Command={  //长度为 64
                    8'h21,//这是一个 1
                    8'd77,
                    8'd40,
                    8'd77,
                    8'd54,
                    songNumColor
                },
                draw2Command={  //长度为 64*5=320
                    8'h21,//这是一个 2
                    8'd72,
                    8'd40,
                    8'd80,
                    8'd40,
                    songNumColor,
                    8'h21,
                    8'd72,
                    8'd40,
                    8'd72,
                    8'd47,
                    songNumColor,
                    8'h21,
                    8'd72,
                    8'd47,
                    8'd80,
                    8'd47,
                    songNumColor,
                    8'h21,
                    8'd80,
                    8'd47,
                    8'd80,
```

```
                8'd54,
                songNumColor,
                8'h21,
                8'd80,
                8'd54,
                8'd72,
                8'd54,
                songNumColor
            };
    parameter drawSuspendingCommand={    //长度为 128
        //这里详细地展示了一个画暂停指令的构成
        8'h21,              //画线指令
        8'd44,              //起始列
        8'd11,              //起始行
        8'd44,              //结束列
        8'd23,              //结束行
        24'hff_ff_ff,       //颜色.这里的 fff 是指白色
        8'h21,              //画线指令
        8'd54,              //起始列
        8'd11,              //起始行
        8'd54,              //结束列
        8'd23,              //结束行
        24'hff_ff_ff        //颜色.这里的 fff 是指白色
    };
    parameter drawPlayingCommand={ //长度为 192
        8'h21,//播放
        8'd44,
        8'd11,
        8'd44,
        8'd23,
        24'hff_ff_ff,
        8'h21,
        8'd44,
        8'd11,
        8'd54,
        8'd17,
        24'hff_ff_ff,
        8'h21,
        8'd44,
        8'd23,
        8'd54,
        8'd17,
        24'hff_ff_ff
    };
```

```
parameter drawBattery={
   8'h22,
   8'd90,
   8'd07,
   8'd95,
   8'd27,
   24'hff_ff_ff,   //边框颜色
   24'hff_ff_ff,    //填充颜色

   8'h22,
  8'd00,
  8'd07,
  8'd05,
  8'd27,
  24'hff_ff_ff,   //边框颜色
  24'hff_ff_ff,    //填充颜色

  8'h22,
     8'd01,
     8'd07,
     8'd04,
     8'd27,
     24'hff_ff_00,   //边框颜色
     24'hff_ff_00,    //填充颜色

      8'h22,
     8'd02,
     8'd07,
     8'd03,
     8'd27,
     24'hff_ff_00,   //边框颜色
     24'hff_ff_00,    //填充颜色

   8'h22,
  8'd91,
  8'd07,
  8'd94,
  8'd22,
  24'h00_ff_00,   //边框颜色
  24'h00_ff_00,    //填充颜色

  8'h22,
    8'd92,
    8'd08,
```

```
        8'd93,
        8'd21,
        24'h00_ff_ff,   //边框颜色
        24'h00_ff_ff,    //填充颜色



      8'h21,              //画线指令
      8'd10,              //起始列
      8'd04,              //起始行
      8'd85,              //结束列
      8'd04,              //结束行
      24'hff_ff_00       //颜色.这里的 fff 是指白色
    };

    parameter drawSongCommand = {
        scrollDeactiveCommand,
        clearLastSongCommand,

        // 长度: 88*2 画一个小兔
        /*
        8'h22,
        8'd01,
        8'd40,
        8'd13,
        8'd54,
        songColor,
        24'h00_00_00,

        /*
        8'h22,
        8'd02,
        8'd54,
        8'd06,
        8'd62,
        songColor,
        24'h00_00_00,   */

        //长度为 920, 画 song 这几个字母的指令
        8'h21,  //字母 s,长度 64*5=320
        8'd16,
        8'd40,
        8'd24,
        8'd40,
```

```
        songColor,
        8'h21,
        8'd24,
        8'd40,
        8'd24,
        8'd47,
        songColor,
        8'h21,
        8'd16,
        8'd47,
        8'd24,
        8'd47,
        songColor,
        8'h21,
        8'd16,
        8'd47,
        8'd16,
        8'd54,
        songColor,
        8'h21,
        8'd16,
        8'd54,
        8'd24,
        8'd54,
        songColor,

        8'h22,  //字母 o,长度 40+48=88
        8'd27,
        8'd40,
        8'd35,
        8'd54,
        songColor,   //边框颜色
        24'h00_00_00,    //填充颜色

        8'h21,  //字母 n,长度 64*3=192
        8'd38,
        8'd40,
        8'd38,
        8'd54,
        songColor,
        8'h21,
        8'd38,
        8'd54,
        8'd46,
```

```
        8'd40,
        songColor,
        8'h21,
        8'd46,
        8'd40,
        8'd46,
        8'd54,
        songColor,

        8'h21,  //字母 G,长度 64*5=320
        8'd49,
        8'd40,
        8'd57,
        8'd40,
        songColor,
        8'h21,
        8'd49,
        8'd40,
        8'd49,
        8'd54,
        songColor,
        8'h21,
        8'd57,
        8'd40,
        8'd57,
        8'd45,
        songColor,
        8'h21,
        8'd49,
        8'd54,
        8'd57,
        8'd54,
        songColor,
        8'h21,
        8'd52,
        8'd45,
        8'd57,
        8'd45,
        songColor
        };
module Divider(
    input I_CLK,    //输入时钟信号，上升沿有效
    input rst,      //同步复位信号，高电平有效
    output reg O_CLK=0    //输出时钟
```

```
);
    parameter defaultMultiple=2;   //默认分频倍数

    integer counter=0;
    always@(posedge I_CLK)
    begin
        if(rst==1)
        begin
            counter<=0;
            O_CLK<=0;
        end
        else
        begin
            if(counter==defaultMultiple/2-1)
            begin
                counter<=0;
                O_CLK<=~O_CLK;
            end
            else
                counter<=counter+1;
        end
    end
endmodule
module mp3 (
    input RESET,
    input MP3_SCLK,         //mp3 的工作时钟
    input MP3_DREQ,

    input [31:0]SongNow,    //当前的歌曲
    input [31:0]Memory,     //当前歌曲的内容
    input [31:0]MusicSize,  //当前歌曲大小
    input IS_SUSPENDING,    //是否暂停
    input [7:0]VOLUME,      //音量大小

    output reg [31:0]MemAddr,       //歌曲地址
    output reg MP3_MOSI,
    output reg MP3_xCS,
    output reg MP3_xDCS,
    output reg MP3_xRSET
);

    //状态机，以及其相应设置
    localparam  hardReset       =7'b1000000,    //硬件复位
                initialisePre   =7'b0100000,    //初始化准备阶段
```

```
                initialise      =7'b0010000,    //初始化
                displayMusic    =7'b0001000,    //播放音乐
                checkState      =7'b0000100,    //检查当前状态
                adjustVolPre    =7'b0000010,    //调整音量的准备阶段
                adjustVol       =7'b0000001;    //调整音量
    reg [6:0]state=hardReset;       //标识当前状态，初始化为硬件复位

    //播放歌曲设置
    reg [31:0]lastSong=32'hff_ff_ff_ff; //记录上一首歌.初始化置为-1

    //模式寄存器设置. 这里使用 reg，方便后续的修改.
    reg[31:0]modeSetting=32'h02_00_08_04;
    reg[31:0]bassSetting=32'h02_02_00_55;
    reg[31:0]volSetting=32'h02_0b_00_00;
    reg[31:0]clockSetting=32'h02_03_98_00;
    reg[127:0]initialSetting;

    //计数器
    integer count=0;                //简单计数器
    integer commandCount=0;          //简单计数器，初始化时使用

    //具体实现
    always@(negedge MP3_SCLK)   //数据在上升沿被采样，所以在下降沿写入
    begin
        case (state)
        hardReset:        //硬件复位
            begin
                MP3_xRSET<=0;
                MP3_xDCS<=1;
                MP3_xCS<=1;
                //状态机转换
                state<=initialisePre;
            end
        initialisePre:    //硬件复位完毕
            begin
                //下一阶段的准备操作
                MP3_xRSET<=1;
                count<=31;
                commandCount<=0;
                initialSetting<={modeSetting,bassSetting,volSetting,cloc
kSetting};
                //状态机转换
                state<=initialise;
            end
```

```
        initialise:       //初始化
            begin
                //配置寄存器. 此过程不受外界信号干扰.
                if(MP3_DREQ)
                begin
                    if(commandCount<4)  //初始化的 4 条指令未发完
                    begin
                        if(count>=0)
                        begin
                            MP3_xCS<=0;
                            MP3_MOSI<=initialSetting[count];
                            count<=count-1;
                        end
                        else    // 特别注意，指令每读 32bit 必须要置一次 xcs 高
                        begin
                            MP3_xCS<=1;
                            count<=31;
                            commandCount<=commandCount+1;
                            initialSetting<=(initialSetting>>32);
                        end
                    end
                    else        //初始化完成
                    begin
                        MP3_xCS<=1;
                        count<=31;      //这里是为播放歌曲而准备的
                        MemAddr<=0;
                        lastSong<=SongNow;  //上一首歌曲切换为当前歌曲
                        //状态机转换
                        state<=checkState; //转到检查播放状态
                    end
                end
                else    //mp3_dreq 为低时，滞空该周期
                    ;
            end
        checkState:       //检查当前状态
            begin
                //状态机转换
                if(RESET)
                    state<=hardReset;
                else if(lastSong!=SongNow)     //歌曲切换
                    state<=initialisePre;      //直接进入初始化预备状态,即
可完成歌曲切换
                else if((volSetting[7:0]!=VOLUME)&&MP3_DREQ) //按下了音量
键
```

```
                    state<=adjustVolPre;        //进入调整音量预备状态
                else if(!IS_SUSPENDING&&MP3_DREQ)   //不处在暂停状态,并且可
以发送数据
                    state<=displayMusic;        //进入播放状态
                else    //否则,该周期滞空
                    ;
            end
        displayMusic:     //播放音乐
            begin
                //发送音频数据
                if(MemAddr>=MusicSize)     //循环播放本首歌曲
                    state<=initialisePre;
                else   //文件未读完
                begin
                    if(count>=0)
                    begin
                        MP3_xDCS<=0;
                        MP3_MOSI<=Memory[count];
                        count<=count-1;
                    end
                    else
                    begin
                        MP3_xDCS<=1;
                        count<=31;
                        MemAddr<=MemAddr+1;
                        //状态机转换
                        state<=checkState;
                    end
                end
            end
        adjustVolPre:     //调整音量的准备阶段
            begin
                count<=31;
                volSetting<={16'h02_0b,VOLUME,VOLUME};      //改变音乐寄存
器的内容
                //状态机转换
                state<=adjustVol;
            end
        adjustVol:        //调整音量
            begin
                if(count>=0)
                begin
                    MP3_xCS<=0;
                    MP3_MOSI<=volSetting[count];
```

```
                    count<=count-1;
                end
                else    // 特别注意，指令每读 32bit 必须要置一次 xcs 高
                begin
                    MP3_xCS<=1;
                    count<=31;
                    //状态机转换
                    state<=checkState;
                end
            end
        default: ;
        endcase
    end

endmodule
module oled(
    input RESET,
    input OLED_CLK,             //oled 工作时钟
    input IS_SUSPENDING,        //是否处在暂停模式


    input [31:0]SongNow,    //当前的歌曲

    output reg OLED_RES,
    output reg OLED_DIN,
    output reg OLED_CS,
    output reg OLED_D_C
);
`include "OLED_CMD.vh"


    //小屏状态机设置
    localparam  reset               =9'b100_000_000,//重置
                initialisePre       =9'b010_000_000,//初始化预备

                initialise          =9'b001_000_000,//初始化

                clearScreen         =9'b000_100_000,//清空窗口
                drawSong            =9'b000_010_000,//画 song 这几个字

                checkState          =9'b000_001_000,//检查状态
```

```
                changeDisplayModePre=9'b000_000_100,//画暂停/播放状态准备
                changeSongNamePre   =9'b000_000_010,//画歌曲名的准备阶段
                sendCommand         =9'b000_000_001;//发送绘图指令
    reg [8:0]state=reset;//初始化为重置状态

    //各类状态信息设置
    reg [31:0]lastSong=32'hff_ff_ff_ff; //记录上一首歌.初始化置-1
    reg lastSuspend=1'b1;   //记录上次的暂停状态

    //各类图形命令以及设置
    localparam  colMax=8'h5f,
                rowMax=8'h3f;   //最大行列地址

    reg [967:0]command;    //指令寄存器, 在状态中会用到

    // localparam drawHelloCommand='h;

    //计数器
    integer count;

    always@(negedge OLED_CLK)   //数据在上升沿被采样, 在下降沿写入
    begin
        case (state)
        reset:
            begin
                OLED_RES<=0;    //重置
                lastSong<=32'hff_ff_ff_ff;
                lastSuspend<=~IS_SUSPENDING;
                //状态机转换
                state<=initialisePre;
```

```
            end
        initialisePre:
            begin
                OLED_RES<=1;    //重置解除
                OLED_CS<=1;
                OLED_D_C<=1;
                count<=303;

                command<={  //长度�?304

                    664'h0,
                    scrollDeactiveCommand,  //首先禁用滚动
                    296'hae_a0_74_a1_00_a2_00_a4_a8_3f_ad_8e_b0_0b_b1_31
_b3_f0_8a_64_8b_78_8c_64_bb_3a_be_3e_87_06_81_91_82_50_83_7d_af
                };

                //状�?�机转换

                state<=initialise;
            end
        initialise:
            begin
                if(count>=0)
                begin
                    OLED_CS<=0;
                    OLED_D_C<=0;
                    OLED_DIN<=command[count];
                    count<=count-1;
                end
                else
                begin
                    OLED_CS<=1;
                    count<=39;

                    //状�?�机转换

                    state<=clearScreen;
                end
            end
        clearScreen:
            begin
                if(count>=0)
                begin
                    OLED_CS<=0;
                    OLED_D_C<=0;
                    OLED_DIN<=clearFullScreenCommand[count];
                    count<=count-1;
```

```
                end
                else
                begin
                    OLED_CS<=1;
                    //状?机转换
                    if(!RESET)  //如果重置信号始终有效，则保持黑?
                    begin
                        count<=599 + 88*7;
                        command<={ //长度?88+512=600
                            //清屏完成?,?要打印上?/下一首键,打印框
框
                            368'h0,
                            drawBlockCommand,
                            drawBattery,
                            //长度 64*8=512，画上?首和下一首的设置
                            8'h21,//上一?
                            8'd12,//?
                            8'd10,
                            8'd12,
                            8'd24,
                            lastNextColor,
                            8'h21,//?
                            8'd24,
                            8'd10,
                            8'd12,
                            8'd17,
                            lastNextColor,
                            8'h21,//?
                            8'd24,
                            8'd24,
                            8'd12,
                            8'd17,
                            lastNextColor,
                            8'h21,//?
```

```
                    8'd24,
                    8'd10,
                    8'd24,
                    8'd24,
                    lastNextColor,


                    8'h21,//下一�?

                    8'd74,//�?
                    8'd10,
                    8'd74,
                    8'd24,
                    lastNextColor,
                    8'h21,//�?
                    8'd74,
                    8'd10,
                    8'd86,
                    8'd17,
                    lastNextColor,
                    8'h21,//�?
                    8'd74,
                    8'd24,
                    8'd86,
                    8'd17,
                    lastNextColor,
                    8'h21,//�?
                    8'd86,
                    8'd10,
                    8'd86,
                    8'd24,
                    lastNextColor
                };
                state<=sendCommand;
            end
            else
                ;
        end
    end

checkState: //�?查状�?
```

```
            begin
                //本处的检查状态设置不同于 MP3.其状态在对应的状态中保持
                if(RESET)  //歌曲改变
                    state<=reset;
                else if(lastSong!=SongNow)
                    state<=changeSongNamePre;

                else if(lastSuspend!=IS_SUSPENDING) //暂停状态改变

                    state<=changeDisplayModePre;
                else
                    ;
            end
        changeSongNamePre:
            begin
                lastSong<=SongNow;

                lastSuspend<=~IS_SUSPENDING;    //暂停状态取反,使得滚
动可以被判断

                count<=1143; //967 + 88*2;
                command <= drawSongCommand;



                //状态机改变

                state<=drawSong;
            end
        drawSong:
            begin
                if(count>=0)
                begin
                    OLED_CS<=0;
                    OLED_D_C<=0;
                    OLED_DIN<=command[count];
                    count<=count-1;
                end
                else
                begin
                    OLED_CS<=1;
                    case (SongNow)
                    0:  begin
                            count<=87;
                            command<={
                                880'h0,
```

```
                        draw0Command
                    };
                end
            1:  begin
                    count<=63;
                    command<={
                        904'h0,
                        draw1Command
                    };
                end
            2:  begin
                    count<=319;
                    command<={
                        648'h0,
                        draw2Command
                    };
                end
            default: ;
            endcase

            //状?机转换

            state<=sendCommand;
        end
    end
changeDisplayModePre:
    begin
        lastSuspend<=IS_SUSPENDING;

        if(IS_SUSPENDING)   //画播?

        begin
            count<=239;
            command<={
                728'h0,
                scrollDeactiveCommand,   //禁用滚动
                clearDisplayCommand,
                drawPlayingCommand
            };
        end
        else
        begin
            count<=223;
            command<={
                744'h0,
                clearDisplayCommand,
```

```
                        drawSuspendingCommand,
                        scrollSettingCommand,
                        scrollActiveCommand    //启用滚动
                    };
                end

                //状态机转换

                state<=sendCommand;
            end
        sendCommand:
            begin
                if(count>=0)
                begin
                    OLED_CS<=0;
                    OLED_D_C<=0;
                    OLED_DIN<=command[count];
                    count<=count-1;
                end
                else
                begin
                    OLED_CS<=1;

                    //状态机转换

                    state<=checkState;
                end
            end
        default: ;
        endcase
    end

endmodule
module top (
    input CLK_100MHz,
    input RESET,        //输入复位信号
    input SUSPEND,      //挂起输出信号


    //mp3按键输入定义

    input MP3_VOL_UP,       //音量加减
    input MP3_VOL_DOWN,
    input MP3_LASTSONG,     //上一曲 下一曲
    input MP3_NEXTSONG,
    input MP3_DREQ,
```

```
    //mp3čž□Í□şçŤŻÍ□Ł
    output MP3_SCLK,
    output MP3_MOSI,
    output MP3_xCS,
    output MP3_xDCS,
    output MP3_xRSET,

    //Í°□Íą□čž□Í□şçŤŻÍ□Ł
    output OLED_CLK,
    output OLED_DIN,
    output OLED_CS,
    output OLED_D_C,
    output OLED_RES,

    //č°□čŻ□çŤŻÍ□Ł
    output IS_SUSPENDING,       //ćŁÍ¤□Í□¨ćš□Í□□ç□ść□?. 1ä¸şćš□Í□?,
0ä¸şć□ć□?
    output IS_SONG0,            //ć□ŻÍ□Śä¸şçŹŹ0éŚ□ć□
    output IS_SONG1,            //ć□ŻÍ□Śä¸şçŹŹ1éŚ□ć□
    output IS_SONG2             //ć□ŻÍ□Śä¸şçŹŹ2éŚ□ć□
);

    //ć□ć□žć□ć□¸čŽžç˝Ž

    //ç□ąäşŽÍ□□Í□é□□Í□śďź□ć□ŹÍ¤□Í□ąčŽžç˝Žä¸□éŚ□ć□ć□¸

    wire [31:0]songNow;           //Í˝□Í□□ć□ć□žçš□ć□ć□?,
Í□□Í?źäťŽ0~songNum-1
    wire [31:0]memAddr;             //ć□ć□¸çš□čŻťć□□é□□, ćŻ□ćŹĄć□-
ć□žć□°ć□ć□¸ć□śç˝?0
    wire [31:0]memory0,memory1,memory2; //Í□ć□žćŻ□éŚ□ć□ć□¸ć□°ć□Žçš□ÍŻ□Í-
□Í□¨
    wire [31:0]music_size=          //Í˝□Í□□ć□ć□¸çš□é□żÍş?,
ćŁ¨ć□□é□?čŚ□ä¸ŽsongNumä¸?ä¸?ÍŻšÍş□
                songNow==0?13164: //50338:   //č□ąčĽżÍ?
                songNow==1?29702:   //ç□Ťçş˘çš□č□¨ć□Ľć□□

                songNow==2?22021:   //55782:   //äş¸ç□ąçš□ć□□äş?

                0;
    wire [31:0]memory=              //Í˝□Í□□ćŁÍ□¨čŻťÍ□□çš□ć□ć□?
                songNow==0?memory0: //č□ąčĽżÍ?
                songNow==1?memory1: //ç□Ťçş˘çš□č□¨ć□Ľć□□

                songNow==2?memory2: //äş¸ç□ąçš□ć□□äş?

                0;
```

```
    assign IS_SONG0=songNow==0?1'b1:0;
    assign IS_SONG1=songNow==1?1'b1:0;
    assign IS_SONG2=songNow==2?1'b1:0;

    //ć□ć□¸é□łé□□čŽžç´Ž
    wire [7:0]volume;

    //ipć ¸č°□ç□?. čż□é□□çš□ćŻ□ä¸Ş blkÍ□□Í□□songNumä¸?ä¸?ÍŻšÍş□
    blk_mem_gen_0 blk_mem_gen_0 (
      .clka(CLK_100MHz),    // input wire clka
      .wea(1'b0),      // input wire [0 : 0] wea
      .addra(memAddr[15:0]),  // input wire [15 : 0] addra
      .dina(0),    // input wire [31 : 0] dina
      .douta(memory0)  // output wire [31 : 0] douta
    );
    blk_mem_gen_1 blk_mem_gen_1 (
      .clka(CLK_100MHz),    // input wire clka
      .wea(1'b0),      // input wire [0 : 0] wea
      .addra(memAddr[14:0]),  // input wire [? : 0] addra
      .dina(0),    // input wire [31 : 0] dina
      .douta(memory1)  // output wire [31 : 0] douta
    );
    blk_mem_gen_2 blk_mem_gen_2 (
      .clka(CLK_100MHz),    // input wire clka
      .wea(1'b0),      // input wire [0 : 0] wea
      .addra(memAddr[16:0]),  // input wire [16 : 0] addra
      .dina(0),    // input wire [31 : 0] dina
      .douta(memory2)  // output wire [31 : 0] douta
    );

    //mp3Í□□é˘□, 100Í□łÍ□Ż
    Divider #(100)
    Divider_mp3(

        .I_CLK(CLK_100MHz),    //čž□Í□Ľć□śé□□äżĄÍ□˘ďź□ä¸□Í□□ć ¸ź□□ć□□

        .rst(1'b0),      //Í□□ćĽÍ¤□ä˝□äżĄÍ□˘ďź□éŤ□ç□ĺÍšłć□□ć□□
        .O_CLK(MP3_SCLK)
    );


    //oledçš□ć□śé□□Í□□é˘□Íż□éĄťćśłÍ¤□Í¤§ďź□Í□ŚÍ□□ć□ ć£□ÍŽ□ć□□Íą□Íš□ć□□ä
˝?
    Divider #(10000)
    Divider_oled(
```

```
    .I_CLK(CLK_100MHz),    //čž▯Í▯Ľć▯śé▯▯äżĄÍ▯ˇďź▯ä¸▯Í▯▯ć¸żć▯▯ć▯▯
    .rst(1'b0),            //Í▯▯ćĽÍ¤▯ä´▯äżĄÍ▯ˇďź▯éŤ▯ç▯ĽÍšłć▯▯ć▯▯
    .O_CLK(OLED_CLK)
);

mp3 mp3(
    .RESET(RESET),
    .MP3_SCLK(MP3_SCLK),           //mp3çš▯Íˇ Ľä´▯ć▯śé▯?
    .MP3_DREQ(MP3_DREQ),

    .SongNow(songNow),             //Í´▯Í▯▯çš▯ć▯ć▯?
    .Memory(memory),               //Í´▯Í▯▯ć▯ć▯¸çš▯Í▯▯ÍŽ?
    .MusicSize(music_size),        //Í´▯Í▯▯ć▯ć▯¸Í¤§Í°▯
    .IS_SUSPENDING(IS_SUSPENDING), //ć▯ŻÍ▯Śćš▯Í▯▯
    .VOLUME(volume),               //é▯łé▯▯Í¤§Í°▯

    .MemAddr(memAddr),             //ć▯ć▯¸Í▯°Í▯▯
    .MP3_MOSI(MP3_MOSI),
    .MP3_xCS(MP3_xCS),
    .MP3_xDCS(MP3_xDCS),
    .MP3_xRSET(MP3_xRSET)
);

oled oled(
    .RESET(RESET),
    .OLED_CLK(OLED_CLK),           //oledÍˇ Ľä´▯ć▯śé▯▯
    .IS_SUSPENDING(IS_SUSPENDING), //ć▯ŻÍ▯ŚÍ¤▯Í▯¨ćš▯Í▯▯ć¨ĄÍź▯
    .SongNow(songNow),             //Í´▯Í▯▯çš▯ć▯ć▯?
    .OLED_RES(OLED_RES),
    .OLED_DIN(OLED_DIN),
    .OLED_CS(OLED_CS),
    .OLED_D_C(OLED_D_C)
);

MusicControl MusicControl(
    .RESET(RESET),
    .MP3_SCLK(MP3_SCLK),
    .SUSPEND(SUSPEND),
    .IS_SUSPENDING(IS_SUSPENDING),
    .MP3_LASTSONG(MP3_LASTSONG),
    .MP3_NEXTSONG(MP3_NEXTSONG),
    .SongNow(songNow),
```

```
        .MP3_VOL_UP(MP3_VOL_UP),        //é□łé□□č°□ć□´
        .MP3_VOL_DOWN(MP3_VOL_DOWN),
        .VOLUME(volume)
    );

endmodule
module top_tb ;
    reg CLK_100MHz;
    reg RESET;          //复位信号
    reg SUSPEND;        //暂停按钮
    //mp3 输入端口
    reg MP3_VOL_UP;         //音量调整
    reg MP3_VOL_DOWN;
    reg MP3_LASTSONG;       //歌曲调整
    reg MP3_NEXTSONG;
    reg MP3_DREQ;
    //mp3 输出端口
    wire MP3_SCLK;
    wire MP3_MOSI;
    wire MP3_xCS;
    wire MP3_xDCS;
    wire MP3_xRSET;
    //小屏输出端口
    wire OLED_CLK;
    wire OLED_DIN;
    wire OLED_CS;
    wire OLED_D_C;
    wire OLED_RES;
    //调试端口
    wire IS_SUSPENDING;         //正处在暂停状态. 1 为暂停; 0 为播放
    wire IS_SONG0;              //是否为第 0 首歌
    wire IS_SONG1;              //是否为第 1 首歌
    wire IS_SONG2;              //是否为第 2 首歌

    initial
    begin
        CLK_100MHz=0;
        RESET=1;
        SUSPEND=0;
        MP3_VOL_UP=0;
        MP3_VOL_DOWN=0;
        MP3_LASTSONG=0;
        MP3_NEXTSONG=0;
        MP3_DREQ=1;
```

```
        #30 RESET=0;
    end
    always
        #2 CLK_100MHz=~CLK_100MHz;

    top top(
        .CLK_100MHz(CLK_100MHz),
        .RESET(RESET),        //复位信号
        .SUSPEND(SUSPEND),      //暂停按钮
        //mp3 输入端口
        .MP3_VOL_UP(MP3_VOL_UP),       //音量调整
        .MP3_VOL_DOWN(MP3_VOL_DOWN),
        .MP3_LASTSONG(MP3_LASTSONG),     //歌曲调整
        .MP3_NEXTSONG(MP3_NEXTSONG),
        .MP3_DREQ(MP3_DREQ),
        //mp3 输出端口
        .MP3_SCLK(MP3_SCLK),
        .MP3_MOSI(MP3_MOSI),
        .MP3_xCS(MP3_xCS),
        .MP3_xDCS(MP3_xDCS),
        .MP3_xRSET(MP3_xRSET),
        //小屏输出端口
        .OLED_CLK(OLED_CLK),
        .OLED_DIN(OLED_DIN),
        .OLED_CS(OLED_CS),
        .OLED_D_C(OLED_D_C),
        .OLED_RES(OLED_RES),
        //调试端口
        .IS_SUSPENDING(IS_SUSPENDING),       //正处在暂停状态. 1 为暂停, 0
为播放
        .IS_SONG0(IS_SONG0),            //是否为第 0 首歌
        .IS_SONG1(IS_SONG1),            //是否为第 1 首歌
        .IS_SONG2(IS_SONG2)             //是否为第 2 首歌
    );
endmodule
`timescale 1ns / 1ps
module OLED_SPI_tb(

    );
    reg CLK,RST,DaCom,Send;
    reg [7:0] DATA;
    wire SCK,SDA,DC,CS,Busy;
```

```
    OLED_SPI OLED_SPI_uut(
    .CLK(CLK),.RST(RST),.DaCom(DaCom),.Send(Send),
    .DATA(DATA),
    .SCK(SCK),.SDA(SDA),.DC(DC),.CS(CS),.Busy(Busy));

    initial begin
    CLK = 0;
    DATA = 8'hA8;
    RST = 1;
    Send = 0;
    DaCom = 0;

    #20 RST = 0; Send = 1;
    #80 DaCom = 1;
    end

    always # 20 CLK = ~CLK;

endmodule

module OLED_tb(

    );
reg CLK,RESET;
reg IS_SUSPENDING;      //是否处在暂停模式
reg [31:0]SongNow;    //当前的歌曲
wire SCK,SDA,DC,CS,Busy;
wire OLED_RES;
wire [7:0]tmp;

OLED OLED_uut(
    .CLK(CLK),.RESET(!RESET),
    .IS_SUSPENDING(0),.SongNow(SongNow),
    .SCK(SCK),.SDA(SDA),.DC(DC),.CS(CS),.Busy(Busy),
    .OLED_RES(OLED_RES),
    .tmp(tmp));

initial begin
CLK <= 0;
RESET <= 1;
SongNow <= 0;
#5 RESET <= 0;
end
always #2 CLK <= !CLK;
```

```
endmodule
```