

同济大学计算机系

数字逻辑课程综合实验报告



学 号 2253666

姓 名 刘禹锡

专 业 计算机科学与技术（精英班）

授课老师 郭玉臣

目录

一、	实验内容	3
二、	2048 数字系统总框图	5
三、	系统控制器设计	6
四、	子系统模块建模	8
五、	测试模块建模	19
六、	实验结果	19
七、	结论.....	21
八、	心得体会及建议	21
九、	附录.....	21

一、实验内容

(一) 系统介绍

项目名称：2048Diu

项目描述：本实验使用 NEXYS A7，实现了一个基于 VGA 显示屏，MP3 模块和七段数码管的 2048 游戏。该数字系统设计时采用自顶向下的设置思路，先将数字系统按照功能，分割成多个子系统，包含 VGA 屏幕显示模块，MP3 播放模块，七段数码管计时，得分显示模块，按键交互等基本功能，此外还完善了按键防抖等细节，数字系统完成度较高，同时调用了前期实验的七段数码管，分频器等模块运用于本次作业中。

游戏玩法：游戏规则同经典的 2048 相同，积分规则为当前屏幕上 16 块方块中数字之和，操作者需按下 N17 按钮重置游戏，此时数码管开始计时，初始得分为 10 分(初始有三个 2 一个 4)表示游戏开始，按下上(M18)下(P18)左(P17)右(M17)可以使方块向相应方向移动，每次有效移动，都会在相反方向产生一个新方块 2，相同数字的方块可以合并成更大的数，以此类推，直到出现 2048 胜利，出现绿色微笑；如果 16 个方框全满，并且 2048 未出现，则游戏失败，出现红色哭脸。另有一种情况，即方块都处于边缘，无需再向按键方向移动。这时计时仍继续，但是分数不变，也不会产生新方块。

游戏运行截图：如下，图一为 VGA 显示效果，此时方块无法移动，表情变为哭脸，数码管停止计时计分，表示游戏结束；图二为硬件连接情况(拆去上面的护板方便按键)，采用片选信号，将 8 个数码管分为两组，左侧显示游戏进行秒数，右侧显示当前得分。



图 1 VGA 显示

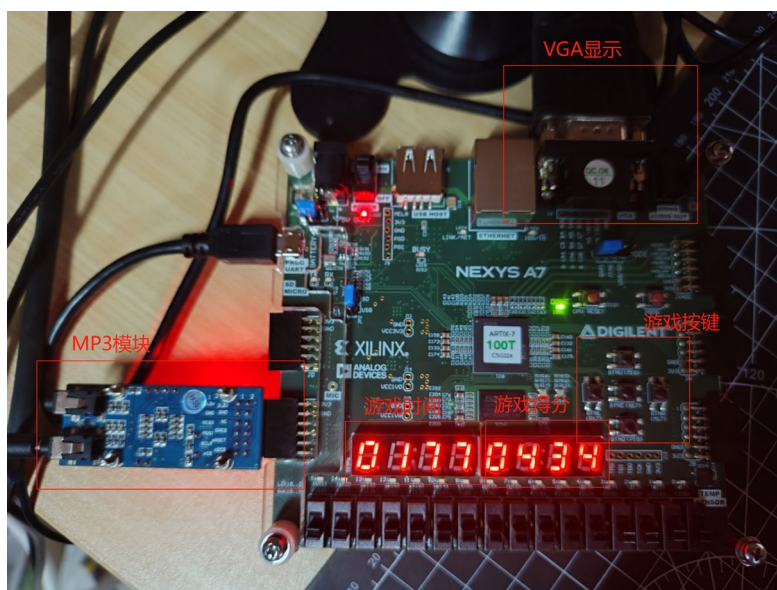


图 2 硬件连接及数码管显示图

(二) 原理介绍

1) VGA 显示功能

我选择了 VGA 显示屏，它的显示原理是利用人眼视觉残留效应，实际上每一个像素点都是依次亮起，从左至右，再从上至下，显示出完整的一张图像。显示器扫描方式为逐行扫描：从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行，电子束回到屏幕的左边下一行的起始位置。每行结束时，用行同步信号进行同步，当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，开始下一帧完成一行扫描的时间称为水平扫描时间，其倒数称为行频率，完成一帧(整屏)扫描的时间称为垂直扫描时间，其倒数称为场频率，即刷新一屏的频率，常见的有 60Hz, 75Hz 等等。标准的 VGA 显示的场频 60Hz, 行频 31.5KHz。本次实验用到的 VGA 场频率为 65MHz。

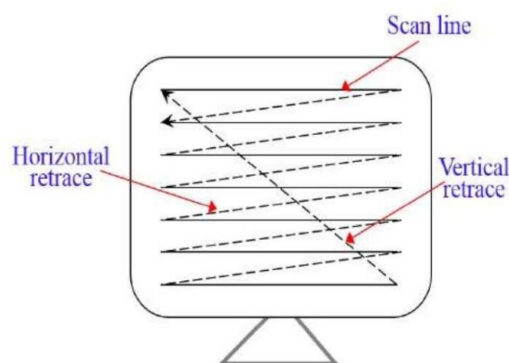


图 2 VGA 扫描示意图

2) MP3 播放功能

使用 VS1003B MP3 模块，通过 SPI 接口与外部控制进行连接，VS1003 通过 7 根信号线共同控制器，分别是：xRSET、XCS、XDCS、SI、SO、SCK 和 DREQ。其中 xRSET 是 VS1003 的复位控制线，低电平有效。DREQ 是数据请求线，用于通知控制器 VS1003 是否可以接收数据。SI(MOSI)、SO(MISO)、SCK 则是 VS1003 的 SPI 通信接口，他们在 XCS 和 XDCS 的控制下执行不同的数据通信。

3) 数码管时间/分数显示

这里基于前期七段数码管的使用，添加了片选，时钟分频等功能，同步接收屏幕数字之和，并采用计数器，对系统 100MHz 时钟进行分频，使得可以同时计时与计分。

4) 按键移动与防抖动

将按键对应的引脚正确分配即可实现按键交互，利用 FIFO 储存结构实现按键除抖；一般的单片机系统，按键作为人机交互工具是必不可少的，但是普通的按键需要消抖处理，极大的增加了程序开销，降低系统实时性。

这里采用 FIFO 按键，无需延时处理消抖，主要流程就是开启一个 10ms 的定时器中断，在中断中扫描按键状态，并对按键状态进行分析消抖处理，如果按键动作，将按键动作压入 FIFO 中，在主循环中读取 FIFO，获取按键状态。

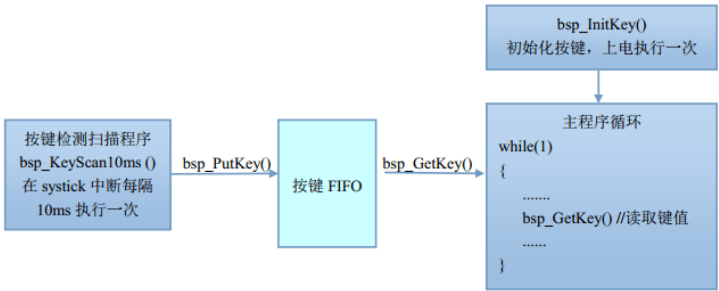


图 3 FIFO 按键消抖

5) 交互性

VGA 显示可以通过编程控制颜色与扫描信号实现，同时通过设计不同数字不同的配色，可以实现颜色的区分；与显示游戏方块同理，只要设计好笑脸显示的颜色与位置，就可以显示笑脸，加上一个哨兵变量即可实现根据游戏状态显示不同的笑脸；重置游戏可以通过重置所有变量来实现。

二、2048 数字系统总框图

(一)总框图

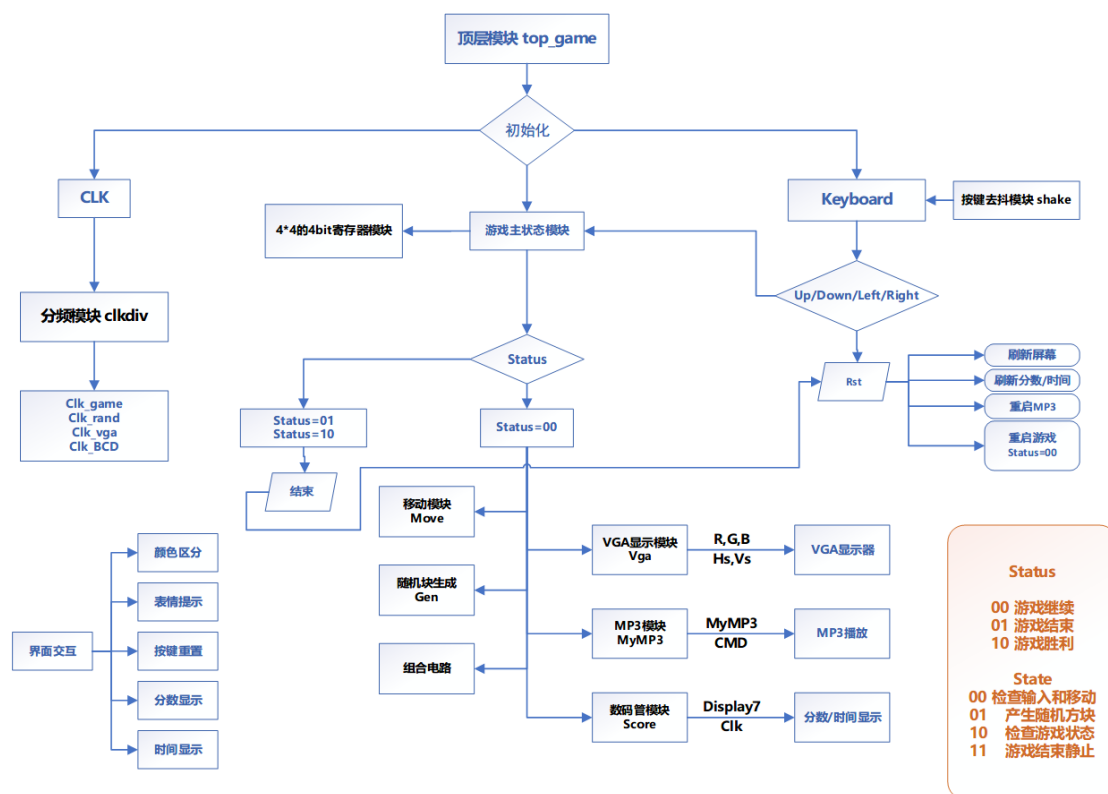


图 4 数字系统总框图

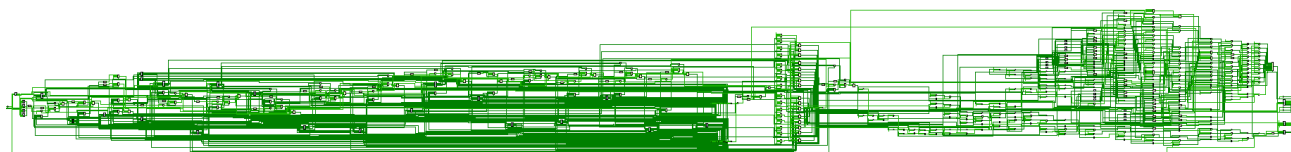


图 6 RTL 实现图

三、系统控制器设计

在数字系统控制器设计过程中，我选择了计数器型控制器的方法。首先，我绘制了一个状态转移图，将数字系统的工作状态划分为四个主要阶段。以下是状态转移图的基本结构：

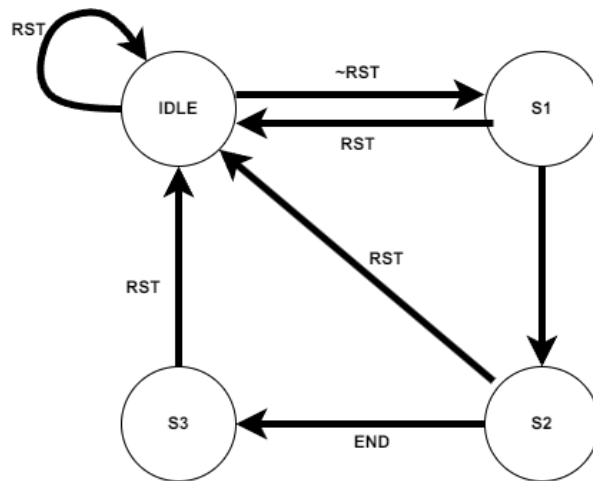


图 7 状态转移图

接着，我对这四个状态进行了编码，使用了一个两位寄存器 `state` 代表系统的状态。具体的状态编码如下：

状态	对应编码	对应状态解释
S0(IDLE)	00	初始状态，即检查输入和移动，游戏继续。
S1	01	判断当前状态，产生随机色块
S2	10	检查游戏状态，结束 or 继续
S3	11	游戏结束，等待重置

表 1 数字系统状态分配表

现态	次态	转换条件
00	00	RST 高电平
	01	RST 低电平且按下移动按键
01	00	RST 高电平
	10	生成新色块
10	00	RST 高电平
	11	16 个区域满或出现 2048
11	00	RST 高电平

图 5 状态转移真值表

激励方程：

```

DFF_next_state_00 = RST & (current_state == 00);
DFF_next_state_01 = ~RST & move_key_pressed & (current_state == 00);
DFF_next_state_10 = RST & (current_state == 01);
DFF_next_state_11 = (current_state == 10) & (area_full | block_2048);
DFF_next_state_00_11 = RST & (current_state == 11);
  
```

控制命令就是 RST 即游戏结束标志 END 等。

在代码中，我使用 `reg [1:0] state` 来表示这些状态，其中 00、01、10、11 分别对应上述四种状态。这种编码方式使得状态之间的转移更加清晰，并且能够有效地满足自启动功能。在数字系统运行的过程中，这四个状态之间不断循环转移，通过一个计数器型控制器实现。这种设计使得系统能够根据不同的状态执行相应的逻辑，从而完成整个数字系统的控制流程。这种状态机的设计能够清晰地表示数字系统的工作流程，便于理解和维护。同时，状态之间的有序转移确保了系统的稳定性和自启动功能。

下面是绘制的控制器 ASM 图：

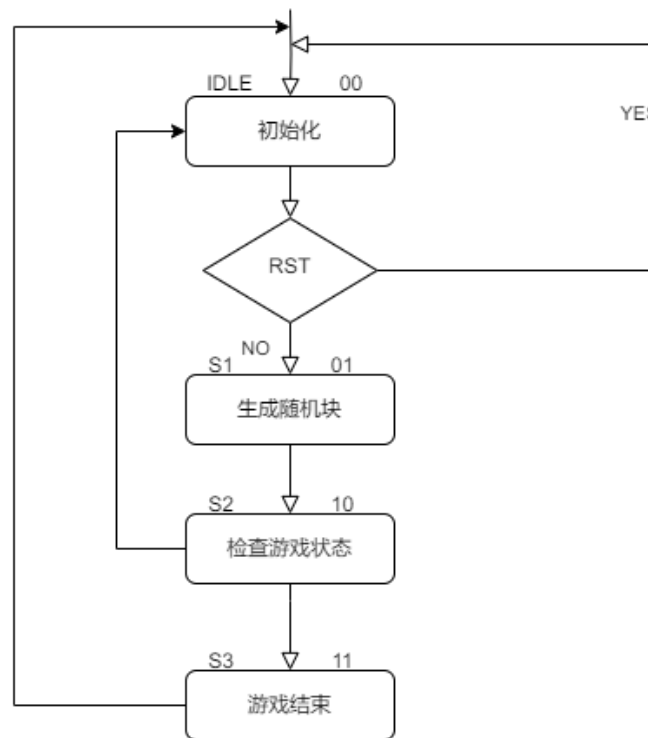


图 6 ASM 流程图

四、子系统模块建模

本数字系统包含时序逻辑和组合逻辑，大致可分为以下几个子系统：

1) 输入子系统：

①按键与去抖模块

通过 FPGA 上的按钮选择游戏重置或要移动的方向，在时钟上升沿对按键输入进行 FIFO 去抖处理，使用一个 32 位 fifo 数组来按键去抖，将处理好的去抖信号赋值给新的 `keyin` 信号，随后对 `keyin` 信号进行条件判断，根据 `keyin` 所代表的按键执行不同的操作，这其中

包含时序逻辑与组合逻辑。

接口定义：

```
module shake(rst,clk,din,dout);  
    input rst,clk; //重置信号，时钟信号，高电平有效  
    input din;    //输入  
    output reg dout; //输出  
    reg [31:0] fifo; //使用 FIFO 防抖
```

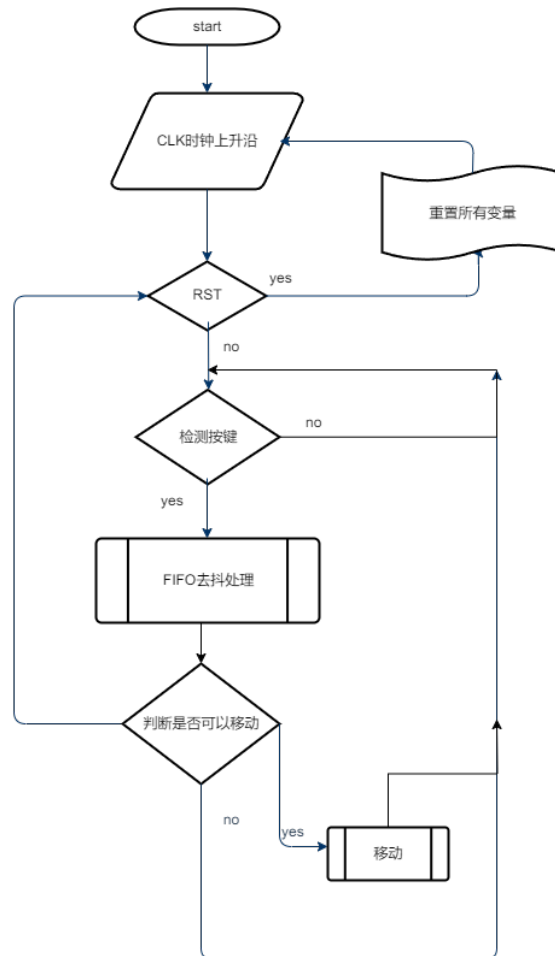


图 7 按键与消抖流程图

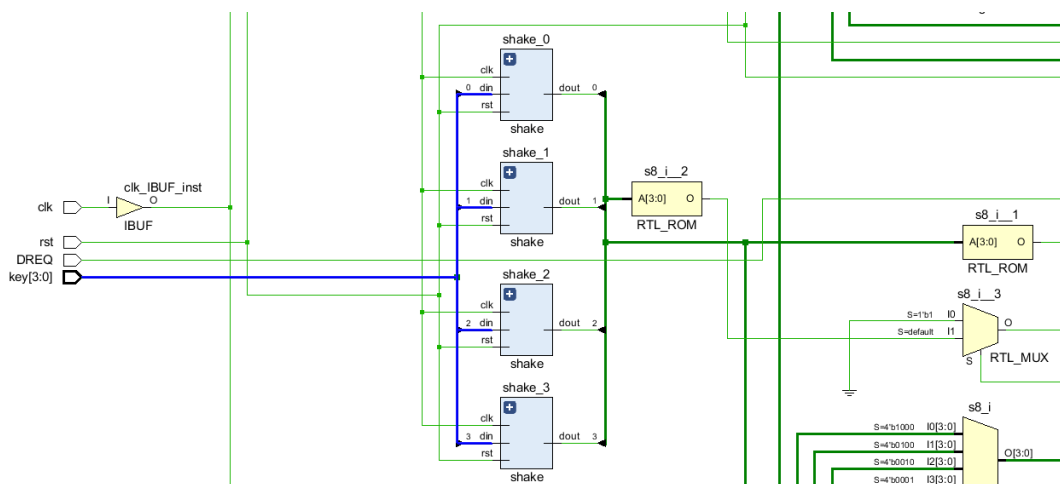


图 8 去抖模块 RTL 图

2) 输出子系统:

①VGA 显示模块

VGA 显示设备通过 VGA 电缆和 N4 开发板相连，主要传输 RGBHV 五种信号：RGB 分别代表 vgaRed, vgaGreen 和 vgaBlue，而 H 代表 Hsync 行同步信号，Vsync 场同步信号。对应的光缆接口如下：

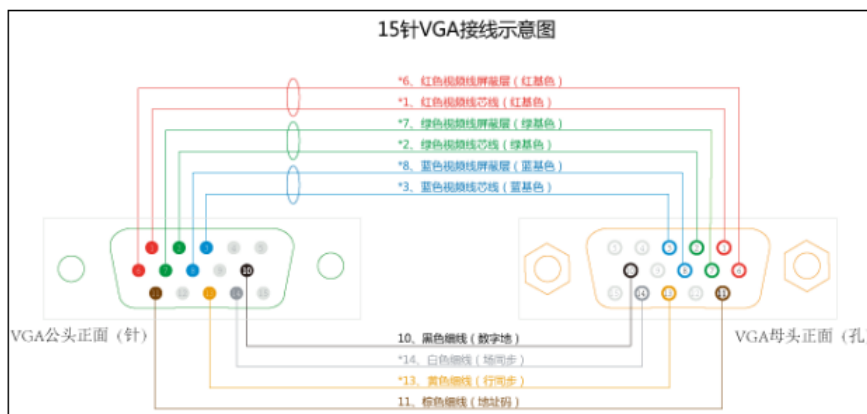


图 9 15 针 VGA 接线示意图

使用同步开关驱动不同像素点颜色的绘制。由于 VGA 必须在 25MHz 时钟频率下工作，需要将系统时钟 CLK 进行 4 分频得到用于信号传输的 clk_vga 信号，通过同步状态下改变 db 值可以切换在该像素点绘制的颜色，flag 用于标记当前色块是否需要更新。

```
assign {r,g,b} = (flag == 1 ? db : 0);
```

接口定义：

```
module vga(rst,dclk,db,r,g,b,hs,vs,x,y);
```

```

input dclk,rst; //时钟信号和重置信号，高电平有效
input [2:0] db; //存储 RGB 信息
output r; //红色
output g; //绿色
output b; //蓝色
output reg hs,vs;
output reg [9:0] x,y;
reg [18:0] addr;
reg [10:0] count_v,count_h;
reg flag;

```

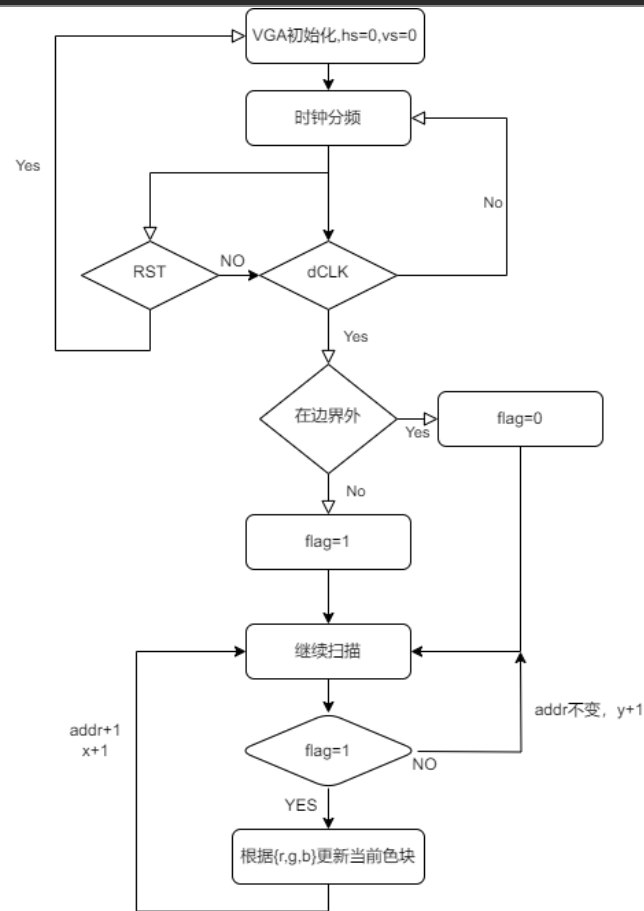


图 10 VGA 状态图

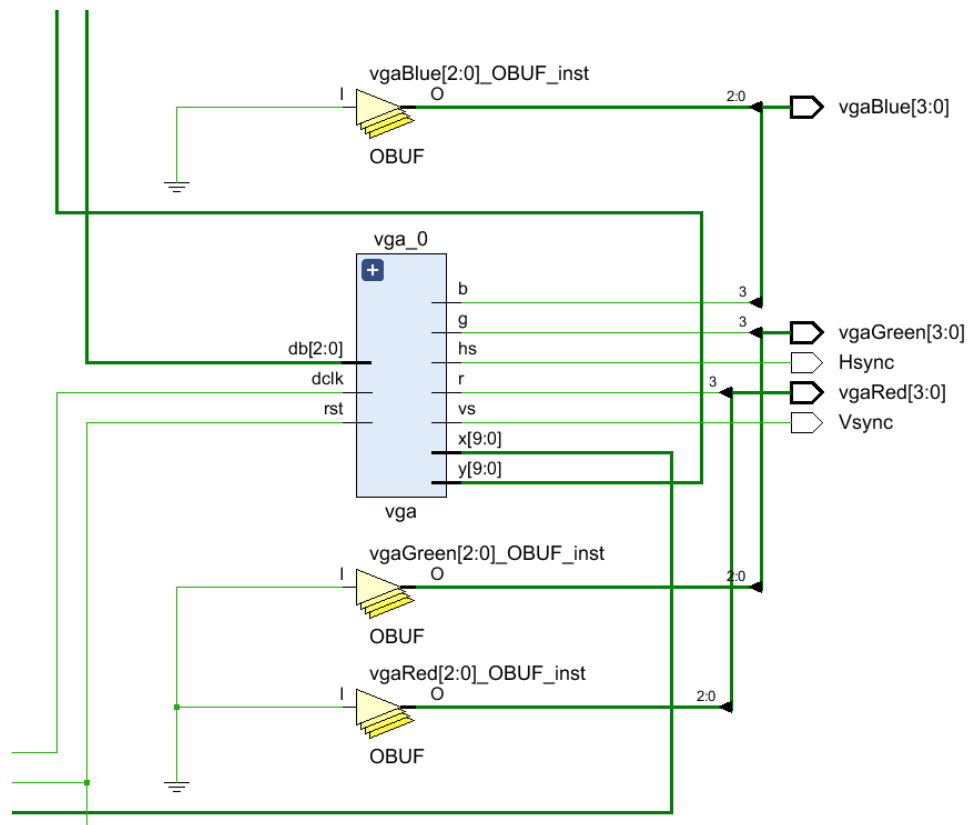


图 11 VGA 模块 RTL 图

②MP3 播放模块

内含时钟分频模块，调用之前的 Divider 对系统时钟进行分频，产生适合 MP3 播放的 1MHz 时钟，对于时钟播放，首先要申请一个 ROM 存放音乐数据，这里使用到了 block_memory_generator IP 核，随后需要将音频数据转化为 COE 文件，对 ROM 进行初始化操作，首先将 wav 音频传入 Matlab 中进行采样，得到 COE 文件，2 进制，十六位，我的 COE 文件大小为 129KB，位宽 16，所以算得读取深度应该为 64500，随后用 COE 文件初始化一个简单单端口 rom，clk_mem_gen_0. 根据需要初始化 MP3 的 SCI 寄存器，SCI 有需要的各种设置。在写入数据时，每 16 位数据送完，置低位 xCS 或 xDCS 要置高，下一个 32 位数据要开始送时又将 xCS 或 xDCS 置低，置低结束后立刻送 32 位数据中的第一个数据。

接口定义：

```
module MyMP3(clk,rst,start,music_id,XRSET,DREQ,XCS,XDCS,SI,SCK);
input clk;           // 系统 100M 时钟
input rst;           // 高电平有效
input start;         // 是否播放
input [2:0] music_id; // 选择的音乐
output reg XRSET;     // 硬件复位，低电平有效
input DREQ;          // 数据请求，高电平有效
```

```

output reg XCS;           // SCI 用于传输命令
output reg XDCS;          // SDI 用来传输数据
output reg SI;             // 向 MP3 写入数据、命令
output reg SCK;            // MP3 时钟

```

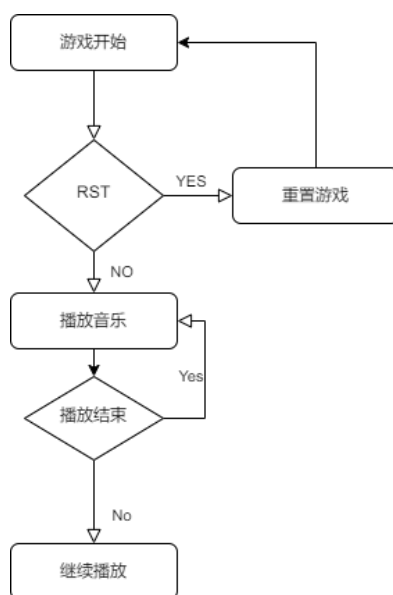


图 12 MP3 播放模块流程图

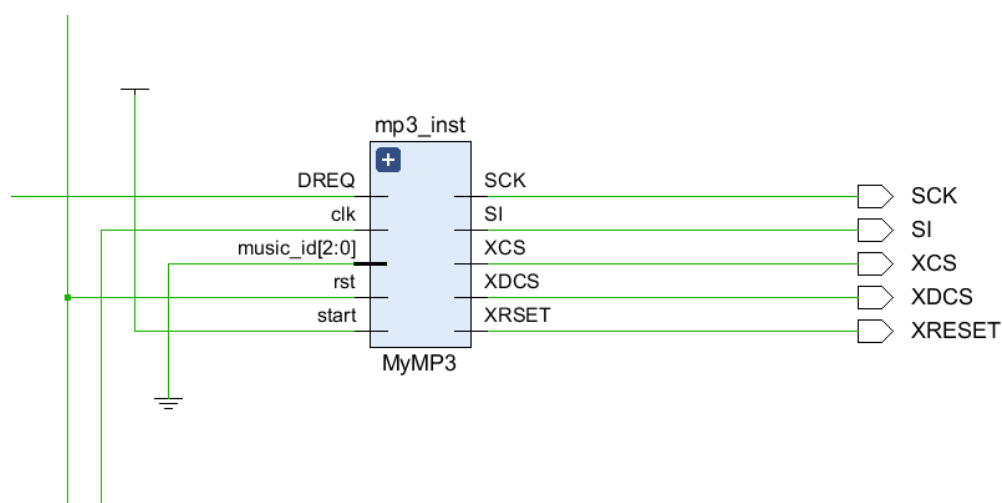


图 13 MyMP3 模块 RTL 图

③数码管分数时间显示模块

使用 FPGA 上的 8 个七段数码管显示分数和时间。在之前的实验中实现了 8 个数码管同步显示的效果，并且可以通过改变时钟分频改变时钟快慢。所以在以往实验基础上，增加了片选信号，将 8 个数码管分为两组，左侧显示时间，即秒数，右侧显示分数。同时为了使计时器单位为秒，且便于人眼观察，需要将 100MHz 的系统时钟分频，经测试，扫描频率在 1000Hz 比较合适，这是因为查找资料发现，多个数码管的连接并不是把每个数码管都独立的与可编程逻辑器件连接，而是把所有的 LED 管的输入连在一起。每次向 LED 写数据时，通过片选选通其中一个 LED，然后把数据写入该 LED 管，因此每个时刻只有一个 LED 管是亮的。所以为了能持续看见 LED 上面的显示内容，必须对 LED 管进行扫描，

即依次并循环地点亮各个 LED 管。利用人眼的视觉暂停效应， 在一定的扫描频率下， 人眼就会看见好几个 LED 一起点亮。扫描频率大小必须合适才能有很好的效果。如果太小，而每个 LED 开启的时间大于人眼的视觉暂停时间， 那么会产生闪烁现象。而扫描频率太大， 则会造成 LED 的频繁开启和关断， 大大增加 LED 功耗（开启和关断的时刻功耗很大）。

在分数显示时，发现，时间虽然是一直在变化，但是分数不是无时无刻变化，所以采用了值传递的方式，屏幕上方框数字总和为总分数，记录在 32 位寄存器 sumScore 中，又因为实例化了 score 模块，sumScore 为其中参数，所以实现了总分的传递，随后将分数和总时间的个位十位百位分割开来，使用片选信号将其在合适位置显示。

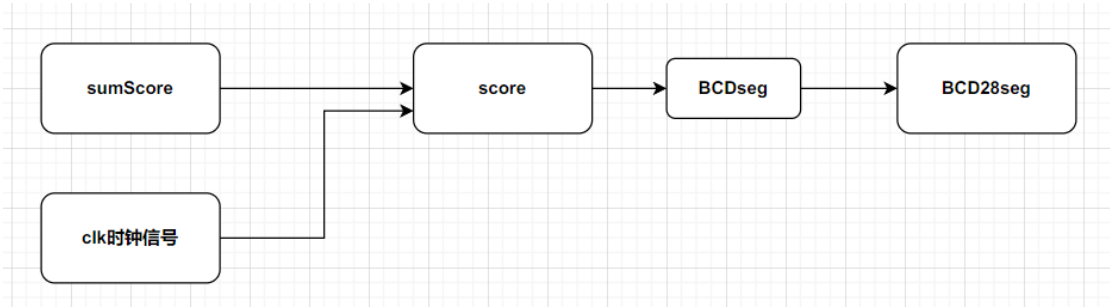
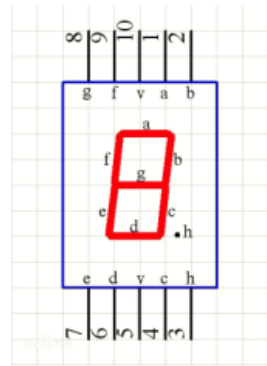


图 14 分数时间显示模块逻辑图

其中数字和模块中 oData 为使能信号，输出值遵循以下规范：



显示数字	oData
0	1000000
1	1111001
2	0100100
3	0110000
4	0011001
5	0010010
6	0000010
7	1111000
8	0000000
9	0010000

图 15 七段数码管显示关系

接口定义：

```
module score(  
    input clock,//时钟信号，同系统  
    input rst,//重置，高电平有效  
    input [1:0] status, //游戏状态  
    input [31:0] sumScore,//总得分  
    output [7:0] an, //使能  
    output [7:0] seg //片选  
);
```

```
module BCDseg(  
    input rst,  
    input clock,  
    input [31:0] score,           //要输出的分数  
    input [31:0] total_time,     //要输出的用时  
    output reg [7:0] seg,        //八位数码管（的某一位）输出  
    output reg [7:0] an          //选择输出为八位数码管其中一个， 低电平有效  
);
```

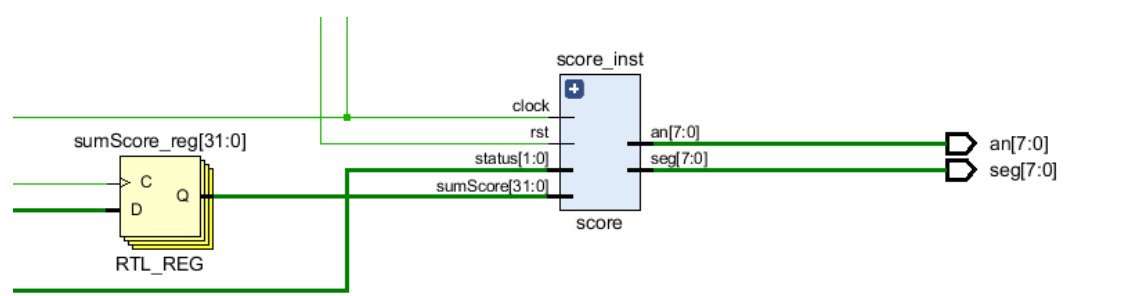


图 16 分数时间显示模块 RTL 图

④色块显示模块

想要在 VGA 屏幕上显示彩色方块，除了使用 ROM/IP 核外，还可以使用硬编码，为每一个方块指定一个颜色，这样就节省了很多内存空间，提升了效率，在 seg 模块中，我将 256 色块数字用 log 形式硬编码保存，这种方法有个优势，就是 2048 游戏都是 2 的指数倍，即 0-2048，所以指定起来也比较方便，这里用一个 288 位的寄存器存放色块颜色，对于颜色的具体选择，我们可以使用一个多路选择器，根据方框存放的数字信息给色块赋值，这样就可以实现色块的颜色存储与显示，节约了空间。

例如：

[illegible]

接口定义:

```
module seg(din,dout);
    input [3:0] din;           //输入， 用于指定数字
    output reg [0:287] dout; //输出， 用于运出硬编码的数字色块
```

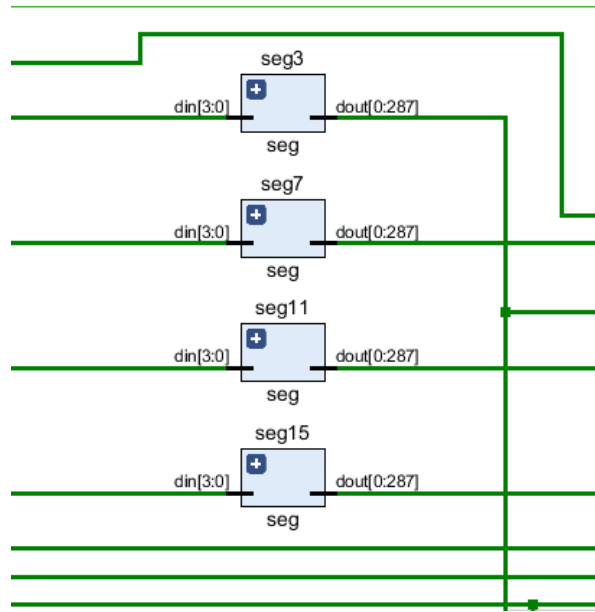


图 17 色块 log 模块 RTL 图

3) 其他子系统:

①时钟分频模块

系统自带 IP 核可以实现时钟分频，但是之前也做过类似的分频器，所以调用了之前的 Divider 模块，整合成 clkdiv 模块用于系统时钟的分频，将 100MHz 的系统时钟分成可供 VGA, MP3, 数码管等使用的时钟信号。

接口定义:

```
module Divider #(parameter num=100000000) //默认分频倍数
(
    input I_CLK, //输入时钟
    output reg O_CLK //输出时钟
);
```

```
module clkdiv(rst,clk,clk_game,clk_rand,clk_vga);
reg [25:0] counter; //计数器
output wire clk_game,clk_rand,clk_vga; //输出的时钟信号
input wire clk,rst; //输入的重置信号和系统时钟信号
```

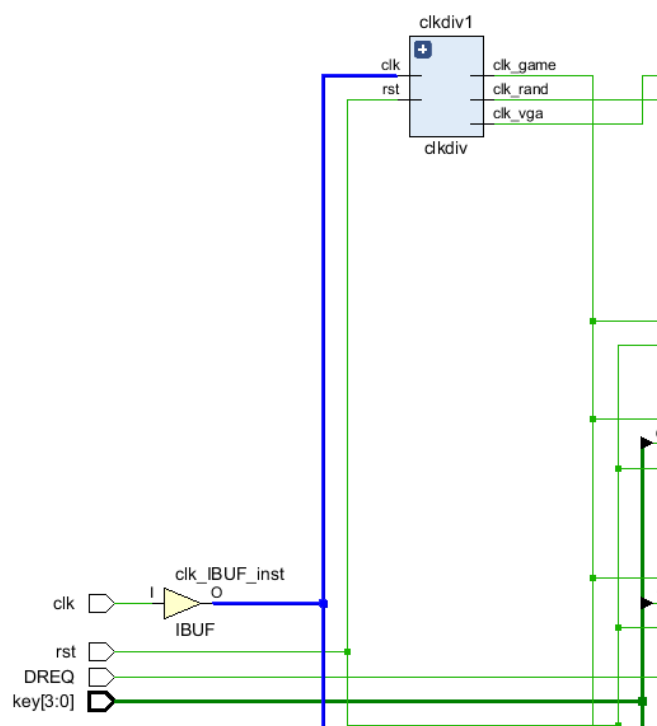



图 18 时钟分频模块 RTL

②移动模块

移动模块需要根据当前的按键状态和当前的游戏状态，生成下一状态的输出。在检测按键状态时，为了避免按键的抖动，我使用了 FIFO 防抖模块处理，并将去抖动后的按键信号存到四位寄存器 keyin 中，作为对移动按键的判断依据，假如判断到有按钮按下，这时应该检测当前状态，如果游戏未结束，那么判断应该移动的方向，具体的状态转移逻辑如下：

如果当前状态的某个方向有按键按下，则在下一状态中该方向为零，其他方向不变。

特殊情况处理：如果连续两个状态的某个方向按键按下，则在下一状态中该方向为零，其他方向不变。如果当前状态和下一状态的某个方向相同，则在下一状态中该方向增加 1。

默认情况：如果以上条件都不满足，下一状态和当前状态相同。

这个过程其实就是检查沿着移动方向的方框情况，直到达到最大可移动的位置，然后将起始位置的数字放到下一位置，实际是更新方框信息，这个过程联系起来就像是在移动。

移动模块：

```
module move(i3,i2,i1,i0,o3,o2,o1,o0);
    input [3:0] i3,i2,i1,i0;          //输入当前状态的信号
    output reg [3:0] o3,o2,o1,o0;    //输出下一状态的信号
```

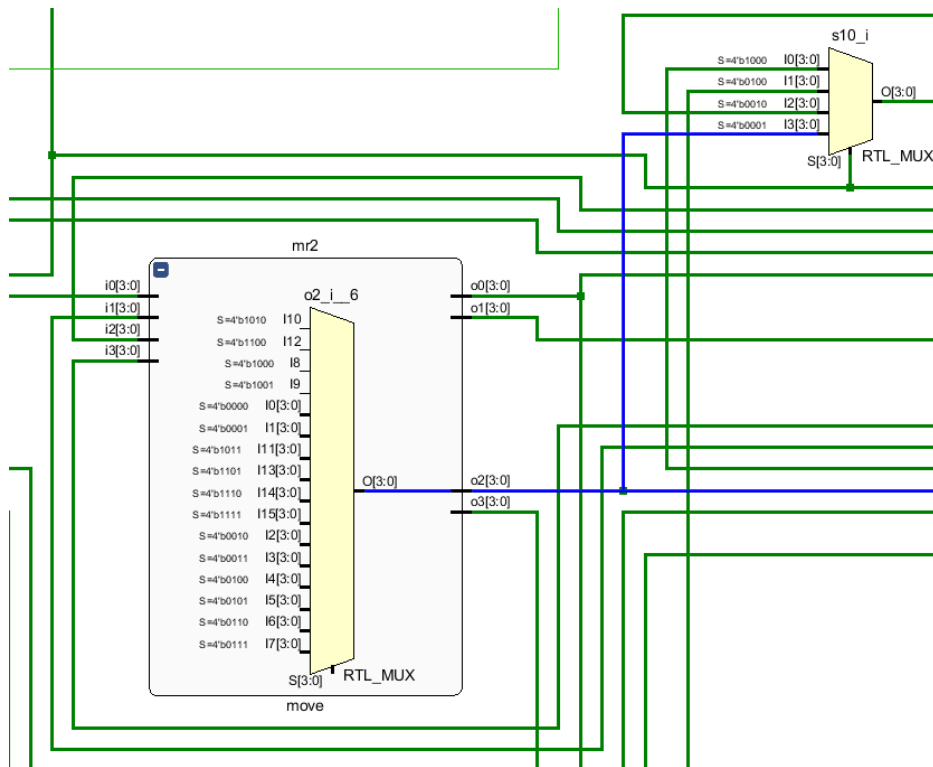


图 19 移动模块 RTL 图

③随机数与随机色块生成模块

本模块用于生成 4 位, 8 位, 16 位的随机数以及在按下按键对应的位置生成一行或一列随机数与三个空位。这是因为 2048 每次有效移动都需要产生一个新的块, 而产生的位置就是与移动方向相反的一行或一列空位, 在这四块中, 我们产生一个随机数和三个空位。Gen 函数同 move 函数相似, 但是要调用 random 用于随机数生成。

接口定义:

```
module random(rst,clk,ran2,ran3,ran4);
    input rst,clk; //时钟信号和重置信号, 高电平有效
    output reg [7:0] ran2; //随机数
    output reg [11:0] ran3; //随机数
    output reg [15:0] ran4; //随机数
```

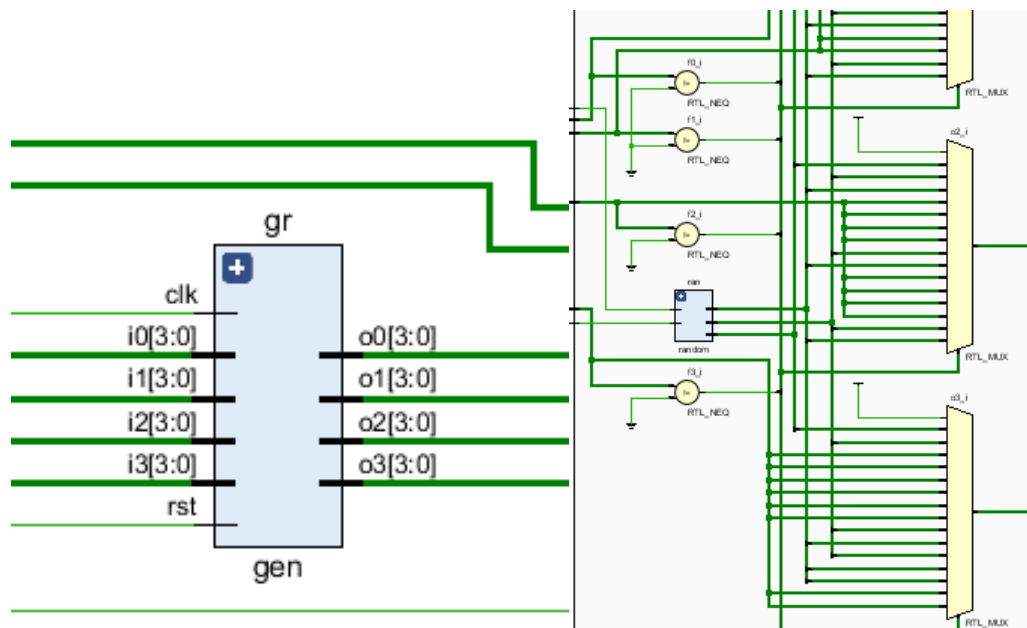


图 20 随机块生成 RTL 图

图 20 random 模块 RTL 图

五、测试模块建模

测试是工程设计中不可或缺的一部分，它可以帮助我们发现许多问题，特别是在这种较为复杂的数字系统中。在本次作业中，我使用了 VGA，MP3，数码管等部件，硬件的开发不同于软件可以通过输出检测，并且 vivado 从综合到实现到生成 bit 流时间特别久，而且有时候还会因为未知错误卡在综合，特别苦恼。对于 verilog 的仿真测试，我们可以使用 modelsim 波形图，对波形进行分析，以及辅助输出等形式，对变量，寄存器进行监控。

在我的设计过程中，主要是 MP3 模块和 VGA 模块遇到了问题，我写了一个 tb_mp3 和 vga 的测试文件，对 mp3 音频播放，即 VGA 信号进行监控。

六、实验结果

波形图仿真：

七、结论

在本实验项目中，我实现了一个简简单单的 2048 小游戏，其能够与 NEXY-A7 板的按键交互，并将结果通过 VGA 扫描显示在屏幕上，同时使用 MP3 播放音乐，数码管显示分数和时间，有较强的游戏性和趣味性。

问题与改进

- ①随机数模块写的有些复杂，在随机的基础上进行改进，使该模块看起来要简练一些。
- ②MP3 播放切换没有搞的很清楚。IP 核中 ROM 的初始化应该按着自己实际的 COE 文件，而不是一味按照教程。
- ③知道了在 verilog 语言里宏定义的定义方法与使用方法与 C 语言里的较大区别，也在这上面犯过不少错误，造成了一定的困扰。在 verilog 中，宏定义的格为：``define [name of the macro] [value of the macro]`宏使用的格式为：``[name of the macro]`一定不能漏了反引号！
- ④学会了用对数形式保存 2 的次幂来简化工作，节省空间。
- ④对于 VGA 的显示原理有了一些了解，看起来复杂，其实只要把握了频率，水平扫描信号，颜色，就能够在屏幕上显示出内容。
- ⑤七段数码管的显示以及时钟分频等采用之前的模块，进行修改，学会了模块复用。

八、心得体会及建议

我选择了 2048 这一经典游戏，使用 VGA 和 MP3 以及数码管这几个部件，采取自顶向下的设计方式，对数字系统进行了模块化设计。对于这种复杂的程序设计，我体会到了功能模块化带来的好处，本程序的几个模块之间除了主模块之外几乎相互独立，极大的方便了调试。

在前期准备时找了很多资料，也测试了一些时间，选择了经典的 2048 游戏，一是出于兴趣，而是游戏数字系统比较规整，更符合数字系统设计这一主题，可以帮助我更好地理解课本知识。

在不断测试代码的过程中发现了很多问题，比如显示时时有时无的彩线，按键去抖的方法，显示游戏区域的乱码等问题，遇到问题后查找资料，使用波形图，辅助输出等进行辅助调试，虽然比较耗时间，但最终还是做出来了。在完成这个项目的过程中，学会了很多东西，也对 Verilog 更加熟悉。体会了一次游戏开发的大部分流程，得到了满足感与成就感，也提高了自信心，这一次的实验项目为我今后的学习与实践提供了宝贵经验。

通过本课程，我学会了从硬件的角度理解问题，学会了软硬件结合。硬件编程与软件编程最大的不同便在于硬件编程不单单需要考虑程序实现的功能，还要考虑编写的代码对应什么电路。比如在本次实验过程中，就多次遇到赋值冲突的问题，在多个 `always` 块中或在同一个 `always` 块中的不同 `if` 语句中对同一个变量进行赋值操作，会使得硬件电路不知道先处理哪个赋值的操作，也不能构建硬件电路。

建议：希望可以采取小组形式，因为有的时候同学们的想法可能比较复杂，作为期末作业又无太多时间去实现很多功能，只能舍去许多想法，如果可以采取小组形式，两人之间可以商量协助，分工明确，相信会让这个大作业的效果更好，对数字系统设计的理解更深，

九、附录

1. 顶层文件 top_game.v

```
/*宏定义*/
`define S16 s15,s14,s13,s12,s11,s10,s9,s8,s7,s6,s5,s4,s3,s2,s1,s0//
日志下每个块的状态
`define U16 u15,u14,u13,u12,u11,u10,u9,u8,u7,u6,u5,u4,u3,u2,u1,u0//
上
`define D16 d15,d14,d13,d12,d11,d10,d9,d8,d7,d6,d5,d4,d3,d2,d1,d0//
下
`define L16 l15,l14,l13,l12,l11,l10,l9,l8,l7,l6,l5,l4,l3,l2,l1,l0//
左
`define R16 r15,r14,r13,r12,r11,r10,r9,r8,r7,r6,r5,r4,r3,r2,r1,r0//
右
`define P16
p15,p14,p13,p12,p11,p10,p9,p8,p7,p6,p5,p4,p3,p2,p1,p0//s16 上一状态
`define b1 (43)
`define b2 (`b1 + 18)
`define b3 (`b2 + 18)
`define b4 (`b3 + 18)
`define b5 (`b4 + 18)
`define t -`b1+18

module
game(rst,clk,key,Hsync,Vsync,vgaRed,vgaGreen,vgaBlue,DREQ,XCS,XDC
S,SCK,SI,XRESET,an,seg); //oData,law
    input rst; //重置信号，高电平有效
    input clk; //时钟信号，系统100MHz 时钟
    //按键相关
    input [3:0] key; //按键输入
    //VGA 相关
    output Hsync,Vsync; //VGA 同步信号
    output [3:0] vgaRed,vgaGreen,vgaBlue; //VGA RGB 信号
    // mp3 相关
    input DREQ; //数据请求，高电平可传输数据
    output XCS; // SCI 传输读写指令
    output XDCS; // SDI 传输数据
    output SCK; // 时钟
    output SI; // 传入 mp3
    output XRESET; // 硬件复位，低电平有效
    // 数码管相关
    output [7:0] an; // 数码管使能
    output [7:0] seg; // 数码管输出

    reg [3:0] `P16,`S16;
    wire [3:0]
```

```

`U16,`D16,`L16,`R16,gu3,gu2,gu1,gu0,gd15,gd14,gd13,gd12,g112,g18,
g14,g10,gr15,gr11,gr7,gr3;//产生随机数 u, d, l, r
reg [15:0] counter;
wire clk_game,clk_rand,clk_vga; //时钟信号
wire [3:0] keyin; //处理的输入
reg [1:0] state; //00 检查输入和移动, 01 产生随机数, 10 检查游戏
状态, 11 结束
reg [7:0] step; //记录步数
reg [31:0] score1;//得分, 测试用
reg [1:0] status; //? `b0 继续, 1 结束, 2 胜利
clkdiv clkdiv1(rst,clk,clk_game,clk_rand,clk_vga); //时钟分频

reg start = 1'b1; //开始标准
//reg over = 1'b0; //结束标志
//assign over = (status==1'b1)?1'b0:1'b1;

//MyMP3 实例化
MyMP3
mp3_inst(.clk(clk), .DREQ(DREQ), .rst(rst), .music_id(0), .XDCS(X
DCS), .XCS(XCS),
.XRSET(XRESET), .SI(SI), .SCK(SCK), .start(start));
//按键防抖实例化
shake shake_3(rst,clk_game,key[3],keyin[3]);
shake shake_2(rst,clk_game,key[2],keyin[2]);
shake shake_1(rst,clk_game,key[1],keyin[1]);
shake shake_0(rst,clk_game,key[0],keyin[0]);
//移动模块实例化
move mu3(s3,s7,s11,s15,u3,u7,u11,u15);
move mu2(s2,s6,s10,s14,u2,u6,u10,u14);
move mu1(s1,s5,s9,s13,u1,u5,u9,u13);
move mu0(s0,s4,s8,s12,u0,u4,u8,u12);
move md3(s15,s11,s7,s3,d15,d11,d7,d3);
move md2(s14,s10,s6,s2,d14,d10,d6,d2);
move md1(s13,s9,s5,s1,d13,d9,d5,d1);
move md0(s12,s8,s4,s0,d12,d8,d4,d0);
move ml3(s12,s13,s14,s15,l12,l13,l14,l15);
move ml2(s8,s9,s10,s11,l8,l9,l10,l11);
move ml1(s4,s5,s6,s7,l4,l5,l6,l7);
move ml0(s0,s1,s2,s3,l0,l1,l2,l3);
move mr3(s15,s14,s13,s12,r15,r14,r13,r12);
move mr2(s11,s10,s9,s8,r11,r10,r9,r8);
move mr1(s7,s6,s5,s4,r7,r6,r5,r4);
move mr0(s3,s2,s1,s0,r3,r2,r1,r0);
//随机数产生模块实例化

```

```

gen gu(rst,clk_rand,s3,s2,s1,s0,gu3,gu2,gu1,gu0);
gen gd(rst,clk_rand,s15,s14,s13,s12,gd15,gd14,gd13,gd12);
gen gl(rst,clk_rand,s12,s8,s4,s0,gl12,gl8,gl4,gl0);
gen gr(rst,clk_rand,s15,s11,s7,s3,gr15,gr11,gr7,gr3);
// 分数时间显示实例化
score
score_inst(.rst(rst),.clock(clk),.an(an),.status(status),.seg(seg),.sumScore(sumScore));

reg [31:0] sumScore; //总得分

always @(posedge clk) begin
    sumScore = 0;
    // 将宏定义展开累加统计得分
    sumScore = sumScore + (s15 ? (1 << s15) : 0);
    sumScore = sumScore + (s14 ? (1 << s14) : 0);
    sumScore = sumScore + (s13 ? (1 << s13) : 0);
    sumScore = sumScore + (s12 ? (1 << s12) : 0);
    sumScore = sumScore + (s11 ? (1 << s11) : 0);
    sumScore = sumScore + (s10 ? (1 << s10) : 0);
    sumScore = sumScore + (s9 ? (1 << s9) : 0);
    sumScore = sumScore + (s8 ? (1 << s8) : 0);
    sumScore = sumScore + (s7 ? (1 << s7) : 0);
    sumScore = sumScore + (s6 ? (1 << s6) : 0);
    sumScore = sumScore + (s5 ? (1 << s5) : 0);
    sumScore = sumScore + (s4 ? (1 << s4) : 0);
    sumScore = sumScore + (s3 ? (1 << s3) : 0);
    sumScore = sumScore + (s2 ? (1 << s2) : 0);
    sumScore = sumScore + (s1 ? (1 << s1) : 0);
    sumScore = sumScore + (s0 ? (1 << s0) : 0);
end

always @(posedge rst or posedge clk_game) begin
    if(rst) begin
        s15 <= 4'h0;s14 <= 4'h1;s13 <= 4'h0;s12 <= 4'h1;
        s11 <= 4'h0;s10 <= 4'h2;s9 <= 4'h0;s8 <= 4'h0;
        s7 <= 4'h0;s6 <= 4'h0;s5 <= 4'h0;s4 <= 4'h0;
        s3 <= 4'h0;s2 <= 4'h0;s1 <= 4'h1;s0 <= 4'h0;
        {`P16} <= 0;step <= 0;score1 <= 0;status <= 0;state <= 0;

    end else begin
        case(state)
            2'b00 : begin
                case(keyin)

```



```

4'b0000 : begin {`S16} <= {`S16};state <= 0;end
4'b1000 : begin {`S16} <= {`U16};state <= 1;end
4'b0100 : begin {`S16} <= {`D16};state <= 1;end
4'b0010 : begin {`S16} <= {`L16};state <= 1;end
4'b0001 : begin {`S16} <= {`R16};state <= 1;end
default : begin {`S16} <= {`S16};state <= 0;end
endcase

{`P16} <= {`S16};step <= step;score1 <= score1;status <=
status;//继续有效 `b1
end

2'b01 : begin
    if({`P16} == {`S16}) begin {`S16} <= {`S16};end
    else case(keyin)
        4'b1000 : begin {`S16} <=
{s15,s14,s13,s12,s11,s10,s9,s8,s7,s6,s5,s4,gu3,gu2,gu1,gu0};end
        4'b0100 : begin {`S16} <=
{gd15,gd14,gd13,gd12,s11,s10,s9,s8,s7,s6,s5,s4,s3,s2,s1,s0};end
        4'b0010 : begin {`S16} <=
{s15,s14,s13,gl12,s11,s10,s9,gl8,s7,s6,s5,gl4,s3,s2,s1,gl0};end
        4'b0001 : begin {`S16} <=
{gr15,s14,s13,s12,gr11,s10,s9,s8,gr7,s6,s5,s4,gr3,s2,s1,s0};end
        default : begin {`S16} <=
{s15,s14,s13,s12,s11,s10,s9,s8,s7,s6,s5,s4,s3,s2,s1,s0};end
    endcase

    {`P16} <= {`P16};step <= step;score1 <= score1;state <=
2;status <= status;
end

2'b10 : begin
    {`S16} <= {`S16};{`P16} <= {`P16};step <= step + 1;score1 <=
score1;

    if({`U16}=={`D16} && {`L16}=={`R16}) begin
        status <= 1;state<= 3;
    end//结束
    else if(keyin == 4'b0000) begin

if(s15==4'hb||s14==4'hb||s13==4'hb||s12==4'hb||s11==4'hb||s10==4'
hb||s9==4'hb||s8==4'hb||s7==4'hb||s6==4'hb||s5==4'hb||s4==4'hb||s
3==4'hb||s2==4'hb||s1==4'hb||s0==4'hb)

        status <= 2;//继续
    else status <= status;

        state <= 0;
    end
end begin
    status <= status;

```

```

        state <= 2;
    end
end
2'b11 : begin {`S16} <= {`S16};{`P16} <= {`P16};step <=
step;score1 <= score1;status <= status;state <= 3;end
    default : begin {`S16} <= {`S16};{`P16} <= {`P16};step <=
step;score1 <= score1;status <= status;state <= 0;end
endcase
end
end

//显示
wire [0:287]
e15,e14,e13,e12,e11,e10,e9,e8,e7,e6,e5,e4,e3,e2,e1,e0;
reg [0:103] faceu,faced;
reg [9:0] position;
reg [7:0] din;
wire [6:0] xin,xin_shift;
wire [2:0] yin;
assign xin = position[6:0];
assign yin = position[9:7];
assign xin_shift = xin + 7'h01;

seg seg15(s15,e15);
seg seg14(s14,e14);
seg seg13(s13,e13);
seg seg12(s12,e12);
seg seg11(s11,e11);
seg seg10(s10,e10);
seg seg9(s9,e9);
seg seg8(s8,e8);
seg seg7(s7,e7);
seg seg6(s6,e6);
seg seg5(s5,e5);
seg seg4(s4,e4);
seg seg3(s3,e3);
seg seg2(s2,e2);
seg seg1(s1,e1);
seg seg0(s0,e0);

reg [2:0] db;
wire [9:0] vx,vy;
wire [7:0] inx;
wire [4:0] iny;

```

```

    wire [2:0] calcy;

    vga
    vga_0(rst,clk_vga,db,vgaRed[3],vgaGreen[3],vgaBlue[3],Hsync,Vsync
    ,vx,vy);

    assign vgaRed[2:0] = 3'h0;
    assign vgaGreen[2:0] = 3'h0;
    assign vgaBlue[2:0] = 3'h0;
    assign inx = vx[9:2]; //横向位置
    assign iny = vy[9:5]; //纵向位置
    assign calcy = vy[4:2]; //白线位置

    always @(inx or iny) begin
        case(iny)

5'h00 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;

            else if(inx<`b1) db <= 3'b000;
            else if(inx<= `b5 && calcy == 3'b000) db <= 3'b111;
            else if(inx<`b2) db <= {3{e15[(inx-`b1)*8+calcy]}} ^
(s15[2:0]);
            else if(inx<`b3) db <= {3{e14[(inx-`b2)*8+calcy]}} ^
(s14[2:0]);
            else if(inx<`b4) db <= {3{e13[(inx-`b3)*8+calcy]}} ^
(s13[2:0]);
            else if(inx<`b5) db <= {3{e12[(inx-`b4)*8+calcy]}} ^
(s12[2:0]);
            else db <= 3'b000;

5'h01 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;

            else if(inx<`b1) db <= 3'b000;
            else if(inx<`b2) db <= {3{e15[(inx+`t)*8+calcy]}} ^
(s15[2:0]);
            else if(inx<`b3) db <= {3{e14[(inx-`b1)*8+calcy]}} ^
(s14[2:0]);
            else if(inx<`b4) db <= {3{e13[(inx-`b2)*8+calcy]}} ^
(s13[2:0]);
            else if(inx<`b5) db <= {3{e12[(inx-`b3)*8+calcy]}} ^
(s12[2:0]);
            else db <= 3'b000;

5'h02 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;

```

```

        else if(inx<`b1) db <= 3'b000;
        else if(inx<=`b5 && calcy == 3'b000) db <= 3'b111;
        else if(inx<`b2) db <= {3{e11[(inx-`b1)*8+calcy]}} ^
(s11[2:0]);
        else if(inx<`b3) db <= {3{e10[(inx-`b2)*8+calcy]}} ^
(s10[2:0]);
        else if(inx<`b4) db <= {3{ e9[(inx-`b3)*8+calcy]}} ^
( s9[2:0]);
        else if(inx<`b5) db <= {3{ e8[(inx-`b4)*8+calcy]}} ^
( s8[2:0]);
        else db <= 3'b000;

5'h03 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;
        else if(inx<`b1) db <= 3'b000;
        else if(inx<`b2) db <= {3{e11[(inx+`t)*8+calcy]}} ^
(s11[2:0]);
        else if(inx<`b3) db <= {3{e10[(inx-`b1)*8+calcy]}} ^
(s10[2:0]);
        else if(inx<`b4) db <= {3{ e9[(inx-`b2)*8+calcy]}} ^
( s9[2:0]);
        else if(inx<`b5) db <= {3{ e8[(inx-`b3)*8+calcy]}} ^
( s8[2:0]);
        else db <= 3'b000;

5'h04 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;
        else if(inx<`b1) db <= 3'b000;
        else if(inx<=`b5 && calcy == 3'b000) db <= 3'b111;
        else if(inx<`b2) db <= {3{ e7[(inx-`b1)*8+calcy]}} ^
(s7[2:0]);
        else if(inx<`b3) db <= {3{ e6[(inx-`b2)*8+calcy]}} ^
(s6[2:0]);
        else if(inx<`b4) db <= {3{ e5[(inx-`b3)*8+calcy]}} ^
(s5[2:0]);
        else if(inx<`b5) db <= {3{ e4[(inx-`b4)*8+calcy]}} ^
(s4[2:0]);
        else db <= 3'b000;

5'h05 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;
        else if(inx<`b1) db <= 3'b000;
        else if(inx<`b2) db <= {3{ e7[(inx+`t)*8+calcy]}} ^
(s7[2:0]);

```

```

        else if(inx<`b3) db <= {3{ e6[(inx-`b1)*8+calcy]}} ^
(s6[2:0]);
        else if(inx<`b4) db <= {3{ e5[(inx-`b2)*8+calcy]}} ^
(s5[2:0]);
        else if(inx<`b5) db <= {3{ e4[(inx-`b3)*8+calcy]}} ^
(s4[2:0]);
        else db <= 3'b000;

5'h06 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;
        else if(inx<`b1) db <= 3'b000;
        else if(inx<=`b5 && calcy == 3'b000) db <= 3'b111;
        else if(inx<`b2) db <= {3{ e3[(inx-`b1)*8+calcy]}} ^
(s3[2:0]);
        else if(inx<`b3) db <= {3{ e2[(inx-`b2)*8+calcy]}} ^
(s2[2:0]);
        else if(inx<`b4) db <= {3{ e1[(inx-`b3)*8+calcy]}} ^
(s1[2:0]);
        else if(inx<`b5) db <= {3{ e0[(inx-`b4)*8+calcy]}} ^
(s0[2:0]);
        else db <= 3'b000;

5'h07 :if(inx==`b1||inx==`b2||inx==`b3||inx==`b4||inx==`b5) db <=
3'b111;
        else if(inx<`b1) db <= 3'b000;
        else if(inx<`b2) db <= {3{ e3[(inx+`t)*8+calcy]}} ^
(s3[2:0]);
        else if(inx<`b3) db <= {3{ e2[(inx-`b1)*8+calcy]}} ^
(s2[2:0]);
        else if(inx<`b4) db <= {3{ e1[(inx-`b2)*8+calcy]}} ^
(s1[2:0]);
        else if(inx<`b5) db <= {3{ e0[(inx-`b3)*8+calcy]}} ^
(s0[2:0]);
        else db <= 3'b000;

5'h08 :if(inx>=`b1 && inx<=`b5 && calcy == 3'b000) db <=
3'b111;
        else db <= 3'b000;

5'h09: if(inx>=72 && inx<=86)begin
        case(status)
            1 : db <= {faceu[(inx-72)*8+calcy],2'b00};
            2 : db <= {1'b0,faceu[(inx-72)*8+calcy],1'b0};
            default : db <= {2'b00,faceu[(inx-72)*8+calcy]};
        endcase
    end

```

```

        else db <= 3'b000;
5'h0a: if(inx>=72 && inx<=86)begin
        case(status)
            1 : db <= {faced[(inx-72)*8+calcy],2'b00};
            2 : db <= {1'b0,faced[(inx-72)*8+calcy],1'b0};
            default : db <= {2'b00,faced[(inx-72)*8+calcy]};
        endcase
    end
    else db <= 3'b000;
    default : db <= 3'b000;
endcase
end

always @(status) begin
    case(status)
        0 : begin faceu <= 104'h08_08_08_08_08_00_00_00_08_08_08_08;
                faced <= 104'h00_00_00_0c_02_01_61_01_02_0c_00_00_00;
            end
        1 : begin faceu <= 104'h22_14_08_14_22_00_00_00_22_14_08_14_22;
                faced <= 104'h00_00_00_1c_22_41_41_41_22_1c_00_00_00;
            end
        2 : begin faceu <= 104'h06_0c_08_08_04_00_00_00_06_0c_08_08_04;
                faced <= 104'h00_00_00_0c_02_01_61_01_02_0c_00_00_00;
            end
        default : begin faceu <=
104'h00_00_00_00_00_00_00_00_00_00_00_00;
                faced <=
104'h00_00_00_0c_02_01_01_01_02_0c_00_00_00;
            end
    endcase
end

endmodule

```

2. 时钟分频 clkdiv.v

```

`timescale 1ns / 1ps
module clkdiv(rst,clk,clk_game,clk_rand,clk_vga);
    reg [25:0] counter;
    output wire clk_game,clk_rand,clk_vga;
    input wire clk,rst;
    assign clk_game = counter[10];
    assign clk_rand = counter[15];
    assign clk_vga = counter[1];

```

```

always @(posedge rst or posedge clk) begin
    if(rst) counter <= 0;
    else counter <= counter + 1;
end
endmodule

```

3. 移动 move.v

```

module move(i3,i2,i1,i0,o3,o2,o1,o0);
    input [3:0] i3,i2,i1,i0;
    output reg [3:0] o3,o2,o1,o0;

    wire f3,f2,f1,f0;
    assign f3 = (i3!=4'h0);
    assign f2 = (i2!=4'h0);
    assign f1 = (i1!=4'h0);
    assign f0 = (i0!=4'h0);

    always @(i3 or i2 or i1 or i0 or f3 or f2 or f1 or f0) begin
        case({f3,f2,f1,f0})
            4'b0000 : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,4'h0};
            4'b0001 : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i0};
            4'b0010 : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i1};
            4'b0011 : if(i1!=i0) {o3,o2,o1,o0} <= {4'h0,4'h0,i1,i0};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i0+4'h1};
            4'b0100 : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i2};
            4'b0101 : if(i2!=i0) {o3,o2,o1,o0} <= {4'h0,4'h0,i2,i0};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i0+4'h1};
            4'b0110 : if(i2!=i1) {o3,o2,o1,o0} <= {4'h0,4'h0,i2,i1};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i1+4'h1};
            4'b0111 : if(i2!=i1 && i1!=i0) {o3,o2,o1,o0} <=
{4'h0,i2,i1,i0};
                       else if(i1==i0) {o3,o2,o1,o0} <=
{4'h0,4'h0,i2,i0+4'h1};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,i1+4'h1,i0};
            4'b1000 : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i3};
            4'b1001 : if(i3!=i0) {o3,o2,o1,o0} <= {4'h0,4'h0,i3,i0};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i0+4'h1};
            4'b1010 : if(i3!=i1) {o3,o2,o1,o0} <= {4'h0,4'h0,i3,i1};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i1+4'h1};
            4'b1011 : if(i3!=i1 && i1!=i0) {o3,o2,o1,o0} <=
{4'h0,i3,i1,i0};
                       else if(i1==i0) {o3,o2,o1,o0} <=
{4'h0,4'h0,i3,i0+4'h1};
                       else {o3,o2,o1,o0} <= {4'h0,4'h0,i1+4'h1,i0};

```

```

        4'b1100 : if(i3!=i2) {o3,o2,o1,o0} <= {4'h0,4'h0,i3,i2};
                else {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,i2+4'h1};
        4'b1101 : if(i3!=i2 && i2!=i0) {o3,o2,o1,o0} <=
{4'h0,i3,i2,i0};
                else if(i2==i0) {o3,o2,o1,o0} <=
{4'h0,4'h0,i3,i0+4'h1};
                else {o3,o2,o1,o0} <= {4'h0,4'h0,i2+4'h1,i0};
        4'b1110 : if(i3!=i2 && i2!=i1) {o3,o2,o1,o0} <=
{4'h0,i3,i2,i1};
                else if(i2==i1) {o3,o2,o1,o0} <=
{4'h0,4'h0,i3,i1+4'h1};
                else {o3,o2,o1,o0} <= {4'h0,4'h0,i2+4'h1,i1};
        4'b1111 : if(i3!=i2 && i2!=i1 && i1!=i0) {o3,o2,o1,o0} <=
{i3,i2,i1,i0};
                else if(i3==i2 && i1==i0) {o3,o2,o1,o0} <=
{4'h0,4'h0,i2+4'h1,i0+4'h1};
                else if(i1==i0) {o3,o2,o1,o0} <=
{4'h0,i3,i2,i0+4'h1};
                else if(i2==i1) {o3,o2,o1,o0} <=
{4'h0,i3,i1+4'h1,i0};
                else {o3,o2,o1,o0} <= {4'h0,i2+4'h1,i1,i0};
        default : {o3,o2,o1,o0} <= {4'h0,4'h0,4'h0,4'h0};
    endcase
end
endmodule

```

4. 随机数生成 gen.v

```

module gen(rst,clk,i3,i2,i1,i0,o3,o2,o1,o0);
    input rst,clk;
    input [3:0] i3,i2,i1,i0;
    output reg [3:0] o3,o2,o1,o0;

    wire f3,f2,f1,f0;
    assign f3=(i3!=4'h0);
    assign f2=(i2!=4'h0);
    assign f1=(i1!=4'h0);
    assign f0=(i0!=4'h0);

    wire [7:0] ran2;
    wire [11:0] ran3;
    wire [15:0] ran4;

    random ran(rst,clk,ran2,ran3,ran4);

```



```

always @(i3 or i2 or i1 or i0 or f3 or f2 or f1 or f0) begin
    case({f3,f2,f1,f0})
        4'b0000 : {o3,o2,o1,o0} <= {ran4};
        4'b0001 : {o3,o2,o1,o0} <= {ran3,i0};
        4'b0010 : {o3,o2,o1,o0} <=
{ran3[11:8],ran3[7:4],i1,ran3[3:0]};
        4'b0011 : {o3,o2,o1,o0} <= {ran2[7:4],ran2[3:0],i1,i0};
        4'b0100 : {o3,o2,o1,o0} <=
{ran3[11:8],i2,ran3[7:4],ran3[3:0]};
        4'b0101 : {o3,o2,o1,o0} <= {ran2[7:4],i2,ran2[3:0],i0};
        4'b0110 : {o3,o2,o1,o0} <= {ran2[7:4],i2,i1,ran2[3:0]};
        4'b0111 : {o3,o2,o1,o0} <= {4'h1,i2,i1,i0};
        4'b1000 : {o3,o2,o1,o0} <= {i3,ran3};
        4'b1001 : {o3,o2,o1,o0} <= {i3,ran2[7:4],ran2[3:0],i0};
        4'b1010 : {o3,o2,o1,o0} <= {i3,ran2[7:4],i1,ran2[3:0]};
        4'b1011 : {o3,o2,o1,o0} <= {i3,4'h1,i1,i0};
        4'b1100 : {o3,o2,o1,o0} <= {i3,i2,ran2[7:4],ran2[3:0]};
        4'b1101 : {o3,o2,o1,o0} <= {i3,i2,4'h1,i0};
        4'b1110 : {o3,o2,o1,o0} <= {i3,i2,i1,4'h1};
        4'b1111 : {o3,o2,o1,o0} <= {i3,i2,i1,i0};
        default : {o3,o2,o1,o0} <= {i3,i2,i1,i0};
    endcase
end
endmodule

```

5. 按键去抖 shake.v

```

`timescale 1ns / 1ps
module shake(rst,clk,din,dout);
    input rst,clk,din;
    output reg dout;
    reg [31:0] fifo; //使用 FIFO 防抖

    always @(posedge rst or posedge clk) begin
        if(rst) fifo <= 0;
        else fifo[31:0] <= {fifo[30:0],din};
    end

    always @(fifo) begin
        if(fifo == 32'h00000000) dout <= 0;
        else if(fifo == 32'hffffffff) dout <= 1;
        else dout <= dout;
    end
end
endmodule

```

6. 方块信息 seg.v

```
module seg(din,dout);  
    input [3:0] din;  
    output reg [0:287] dout;  
  
    always @(din) begin  
        case(din)//显示颜色  
            4'h0                                :                dout                    <=  
288'h00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_  
00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00;//0  
            4'h1                                :                dout                    <=  
288'h00_00_00_00_00_00_00_00_02_02_03_00_00_00_00_00_00_00_00_00_  
00_00_00_00_00_00_e0_a0_a0_00_00_00_00_00_00_00_00;//2  
            4'h2                                :                dout                    <=  
288'h00_00_00_00_00_00_00_00_03_00_03_00_00_00_00_00_00_00_00_00_  
00_00_00_00_00_00_80_80_e0_00_00_00_00_00_00_00_00_00;//4  
            4'h3                                :                dout                    <=  
288'h00_00_00_00_00_00_00_00_03_02_03_00_00_00_00_00_00_00_00_00_  
00_00_00_00_00_00_e0_a0_e0_00_00_00_00_00_00_00_00;//8  
            4'h4                                :                dout                    <=  
288'h00_00_00_00_00_00_00_00_03_00_00_03_02_02_00_00_00_00_00_00_  
00_00_00_00_00_e0_00_00_e0_a0_e0_00_00_00_00_00_00_00;//16  
            4'h5                                :                dout                    <=  
288'h00_00_00_00_00_00_00_02_02_03_00_02_02_03_00_00_00_00_00_00_  
00_00_00_00_a0_a0_e0_00_e0_a0_a0_00_00_00_00_00_00_00;//32  
            4'h6                                :                dout                    <=  
288'h00_00_00_00_00_00_00_03_02_02_00_03_00_03_00_00_00_00_00_00_  
00_00_00_00_e0_a0_e0_00_80_80_e0_00_00_00_00_00_00_00;//64  
            4'h7                                :                dout                    <=  
288'h00_00_00_00_00_00_03_00_00_02_02_03_00_03_02_03_00_00_00_00_  
00_00_00_e0_00_00_e0_a0_a0_00_e0_a0_e0_00_00_00_00_00;//128  
            4'h8                                :                dout                    <=  
288'h00_00_00_00_02_02_03_00_03_02_02_00_03_02_02_00_00_00_00_00_  
00_00_e0_a0_a0_00_a0_a0_e0_00_e0_a0_e0_00_00_00_00_00;//256  
            4'h9                                :                dout                    <=  
288'h00_00_00_00_03_02_02_00_00_03_00_00_02_02_03_00_00_00_00_00_  
00_00_a0_a0_e0_00_00_e0_00_00_e0_a0_a0_00_00_00_00_00;//512  
            4'ha                                :                dout                    <=  
288'h00_00_00_03_00_00_03_02_03_00_02_02_03_00_03_00_03_00_00_00_  
00_e0_00_00_e0_20_e0_00_e0_a0_a0_00_80_80_e0_00_00_00_00;//1024  
            4'hb                                :                dout                    <=  
288'h00_00_02_02_03_00_03_02_03_00_03_00_03_00_03_02_03_00_00_00_  
e0_a0_a0_00_e0_20_e0_00_80_80_e0_00_e0_a0_e0_00_00_00_00;//2048  
            4'hc                                :                dout                    <=
```

```

288'h00_00_03_00_03_00_03_02_03_00_03_02_03_00_03_02_02_00_00_00_
80_80_e0_00_e0_20_e0_00_a0_a0_e0_00_e0_a0_e0_00; //4096
    4'hd          :          dout          <=
288'h00_00_03_02_03_00_00_03_00_00_03_02_03_00_02_02_03_00_00_00_
e0_a0_e0_00_00_e0_00_00_a0_a0_e0_00_e0_a0_a0_00; //8192
    default:
dout<=288'hff_80_80_80_80_80_80_80_80_80_80_80_80_80_80_80_80_80_
ff_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00_00; //none
    endcase
end
endmodule

```

7. MP3 模块 MyMP3.v

```

`timescale 1ns / 1ps

module MyMP3(clk,rst,start,music_id,XRSET,DREQ,XCS,XDCS,SI,SCK);
input clk;          // 系统 100M 时钟
input rst;          // 高电平有效
input start;        // 是否播放
input [2:0] music_id; // 选择的音乐

output reg XRSET;    // 硬件复位，低电平有效
input DREQ;          // 数据请求，高电平有效
output reg XCS;      // SCI 用于传输命令
output reg XDCS;     // SDI 用来传输数据
output reg SI;       // 向 MP3 写入数据、命令
output reg SCK;      // MP3 时钟

reg init = 0;
// 1M 时钟
wire clk_1M;
Divider #(100) clk_mp3(clk, clk_1M);

// IP 核实例化
reg[11:0] addr;
wire [15:0] get_data;
reg [15:0] music_data;
blk_mem_gen_0 music_0 (.clka(clk),.ena(1),.addra({music_id,
addr}),.douta(get_data));

reg [3:0] condition = 0; // 状态
reg [63:0] cmd = {32'h02000804, 32'h020B8080}; // 要写入的命令

//// 修改音量的例子，假设你要将音量调大到最大值
//reg [15:0] new_volume = 16'hFFFF; // 设置新的音量值

```

```

///// 在适当的时机将新的音量值写入 cmd
//always @(posedge clk) begin
//    cmd[15:0] <= new_volume; // 将新的音量值写入 cmd 的低 16 位
//end

```

```

// 变量
integer cnt = 0;           // 计数器
integer num = 0;           // 记录进度

```

```

// 状态常量
parameter DELAY = 1;
parameter PRE_CMD = 2;
parameter WRITE_CMD = 3;
parameter PRE_DATA = 4;
parameter WRITE_DATA = 5;
// 延时常量
parameter DELAY_TIME = 500000;

```

```

always @(posedge clk_1M)
begin
    if(rst || !init) begin                // 进行初始化
        init <= 1;
        XRSET <= 0;
        SCK <= 0;
        XCS <= 1;                        //禁止写入命令
        XDCS <= 1;                        //禁止将数据传入 MP3
        condition <= DELAY;
        addr <= 0;
        cnt <= 0;
    end
    else begin
        if(start) begin
            case (condition)
                DELAY: begin
                    if(cnt == DELAY_TIME) begin // 延时结束
                        cnt <= 0;
                        condition <= PRE_CMD;
                        XRSET <= 1;                // 硬复位
                    end
                    else begin                    // 等待延时
                        cnt <= cnt + 1;
                    end
                end
            endcase
        end
    end
end

```

```

end
PRE_CMD: begin
    SCK <= 0; // MP3 时钟下降沿，更新数
据
    if(num == 2) begin // 写入命令完成
        condition <= PRE_DATA;
        num <= 0;
    end
    else begin
        if(DREQ) begin
            cnt <= 0;
            condition <= WRITE_CMD;
        end
    end
end
WRITE_CMD: begin
    if(DREQ) begin
        if(clk) begin
            if(cnt == 32) begin // 配置寄存器命令完毕
                cnt <= 0; // 重置计数器
                XCS <= 1; //禁止写入命令
                condition <= PRE_CMD;
                num <= num + 1;
            end
            else begin
                XCS <= 0; //允许写入命令
                SI <= cmd[63]; // 写入命令，更新
                cmd <= {cmd[62: 0], cmd[63]}; // 命令移位
                cnt <= cnt + 1;
            end
        end
    end
    SCK <= ~SCK; // 交替变换，进行数据更新与采样
end
PRE_DATA: begin
    if(DREQ) begin
        SCK <= 0;
        condition <= WRITE_DATA;
        music_data <= get_data; // 音乐数据传递，方便移位
        cnt <= 0;
    end
end
WRITE_DATA: begin
    if(SCK) begin

```

```

        if(cnt == 16) begin                // 结束一次 2 字节数据写入，进
行下一次数据的准备、更新
            XDCS <= 1;                    // 禁止将数据传入 MP3
            addr <= addr + 1;              // ROM 地址增加
            cnt <= 0;
            condition <= PRE_DATA;
        end
        else begin                          // 进行音乐数据的写
入
            XDCS <= 0;                    // 允许将数据传入 MP3
            SI <= music_data[15];         // 写入音乐数据
            music_data <= {music_data[14:0],music_data[15]};
            cnt <= cnt + 1;
        end
    end
    SCK <= ~SCK;    // 数据更新与采样
end
default: ;
endcase
end
end
end
endmodule

```

8. 管脚约束文件 2048Diu.xdc

```

# 时钟信号约束
set_property -dict { PACKAGE_PIN E3 IOSTANDARD LVCMOS33 } [get_ports
{ clk }];
# VGA 信号约束
set_property -dict { PACKAGE_PIN A6 IOSTANDARD LVCMOS33 } [get_ports
{ vgaGreen[3] }];
set_property -dict { PACKAGE_PIN B6 IOSTANDARD LVCMOS33 } [get_ports
{ vgaGreen[2] }];
set_property -dict { PACKAGE_PIN A5 IOSTANDARD LVCMOS33 } [get_ports
{ vgaGreen[1] }];
set_property -dict { PACKAGE_PIN C6 IOSTANDARD LVCMOS33 } [get_ports
{ vgaGreen[0] }];

set_property -dict { PACKAGE_PIN A4 IOSTANDARD LVCMOS33 } [get_ports
{ vgaRed[3] }];
set_property -dict { PACKAGE_PIN C5 IOSTANDARD LVCMOS33 } [get_ports
{ vgaRed[2] }];
set_property -dict { PACKAGE_PIN B4 IOSTANDARD LVCMOS33 } [get_ports

```

```

{ vgaRed[1] }];
set_property -dict { PACKAGE_PIN A3 IOSTANDARD LVCMOS33 } [get_ports
{ vgaRed[0] }];

set_property -dict { PACKAGE_PIN D8 IOSTANDARD LVCMOS33 } [get_ports
{ vgaBlue[3] }];
set_property -dict { PACKAGE_PIN D7 IOSTANDARD LVCMOS33 } [get_ports
{ vgaBlue[2] }];
set_property -dict { PACKAGE_PIN C7 IOSTANDARD LVCMOS33 } [get_ports
{ vgaBlue[1] }];
set_property -dict { PACKAGE_PIN B7 IOSTANDARD LVCMOS33 } [get_ports
{ vgaBlue[0] }];

# VGA 同步信号约束
set_property -dict { PACKAGE_PIN B11 IOSTANDARD LVCMOS33 } [get_ports
{ Hsync }];
set_property -dict { PACKAGE_PIN B12 IOSTANDARD LVCMOS33 } [get_ports
{ Vsync }];

# 复位信号约束
set_property -dict { PACKAGE_PIN N17 IOSTANDARD LVCMOS33 } [get_ports
{ rst }];

# 按键信号约束
set_property -dict { PACKAGE_PIN M18 IOSTANDARD LVCMOS33 } [get_ports
{ key[3] }];
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports
{ key[2] }];
set_property -dict { PACKAGE_PIN P17 IOSTANDARD LVCMOS33 } [get_ports
{ key[1] }];
set_property -dict { PACKAGE_PIN M17 IOSTANDARD LVCMOS33 } [get_ports
{ key[0] }];

# MP3 信号约束
set_property IOSTANDARD LVCMOS33 [get_ports XCS]
set_property IOSTANDARD LVCMOS33 [get_ports XDCS]
set_property IOSTANDARD LVCMOS33 [get_ports XRESET]
set_property IOSTANDARD LVCMOS33 [get_ports SCK]
set_property IOSTANDARD LVCMOS33 [get_ports SI]
set_property IOSTANDARD LVCMOS33 [get_ports DREQ]
set_property IOSTANDARD LVCMOS33 [get_ports start]
set_property IOSTANDARD LVCMOS33 [get_ports over]

set_property PACKAGE_PIN J2 [get_ports DREQ]

```

```

set_property PACKAGE_PIN F6 [get_ports XRESET]
set_property PACKAGE_PIN K1 [get_ports XDCS]
set_property PACKAGE_PIN E7 [get_ports XCS]
set_property PACKAGE_PIN J4 [get_ports SI]
set_property PACKAGE_PIN J3 [get_ports SCK]
set_property PACKAGE_PIN H17 [get_ports start]
set_property PACKAGE_PIN V11 [get_ports over]

# 数码管信号约束

##7 segment display

set_property -dict { PACKAGE_PIN T10 IOSTANDARD LVCMOS33 }
[get_ports { seg[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10 IOSTANDARD LVCMOS33 }
[get_ports { seg[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16 IOSTANDARD LVCMOS33 }
[get_ports { seg[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13 IOSTANDARD LVCMOS33 }
[get_ports { seg[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15 IOSTANDARD LVCMOS33 }
[get_ports { seg[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11 IOSTANDARD LVCMOS33 }
[get_ports { seg[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18 IOSTANDARD LVCMOS33 }
[get_ports { seg[6] }]; #IO_L4P_T0_D04_14 Sch=cg

set_property -dict { PACKAGE_PIN H15 IOSTANDARD LVCMOS33 }
[get_ports { seg[7] }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17 IOSTANDARD LVCMOS33 }
[get_ports { an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18 IOSTANDARD LVCMOS33 }
[get_ports { an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9 IOSTANDARD LVCMOS33 }
[get_ports { an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14 IOSTANDARD LVCMOS33 }
[get_ports { an[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14 IOSTANDARD LVCMOS33 }
[get_ports { an[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14 IOSTANDARD LVCMOS33 }
[get_ports { an[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2 IOSTANDARD LVCMOS33 }
[get_ports { an[6] }]; #IO_L23P_T3_35 Sch=an[6]

```



```
set_property -dict { PACKAGE_PIN U13      IOSTANDARD LVCMOS33 }
[get_ports { an[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]
```

9. Testbench 文件

```
`timescale 1ns / 1ps
module game_tb;
    // Parameters
    parameter CLK_PERIOD = 10; // Clock period in nanoseconds
    // Inputs
    reg rst;
    reg clk;
    reg [3:0] key;
    reg DREQ;
    reg XCS;
    reg XDCS;
    reg SCK;
    reg SI;
    reg XRESET;
    // Outputs
    wire Hsync;
    wire Vsync;
    wire [3:0] vgaRed;
    wire [3:0] vgaGreen;
    wire [3:0] vgaBlue;
    wire [7:0] an;
    wire [7:0] seg;
    // Instantiate the game module
    game uut (
        .rst(rst),
        .clk(clk),
        .key(key),
        .Hsync(Hsync),
        .Vsync(Vsync),
        .vgaRed(vgaRed),
        .vgaGreen(vgaGreen),
        .vgaBlue(vgaBlue),
        .DREQ(DREQ),
        .XCS(XCS),
        .XDCS(XDCS),
        .SCK(SCK),
        .SI(SI),
        .XRESET(XRESET),
```

```

        .an(an),
        .seg(seg)
    );
    // Instantiate the score module
    score score_inst (
        .rst(rst),
        .clock(clk),
        .an(an),
        .status(status),
        .seg(seg),
        .sumScore(sumScore)
    );
    // Clock Generation
    always begin
        #CLK_PERIOD/2 clk = ~clk;
    end
    // Initial block
    initial begin
        // Initialize inputs
        rst = 1;
        key = 4'b0000;
        DREQ = 0;
        XCS = 0;
        XDCS = 0;
        SCK = 0;
        SI = 0;
        XRESET = 1;
        // Apply reset
        #20 rst = 0;
        // Run simulation for some time
        #1000;
        // End simulation
        $stop;
    end
endmodule

```