XXXX大学

数字电路与逻辑设计实验报告

**砸彩蛋游戏机的设计与实现**

院系：

班级：

姓名：

学号：

指导老师：

2020年11月

摘要

本次实验课本次实验课题是用硬件描述语言 Verilog来设计实现砸彩蛋游戏机，其基本功能为砸彩蛋游戏。在砸彩蛋前有设备自检功能。游戏过程通过点阵显示彩蛋，使用键盘按键模拟砸蛋，通过数码管显示倒计时和得分。自检，游戏等待，游戏结束及游戏中砸中彩蛋均有音乐提示。另可通过按键，切换彩蛋显示时间，调整游戏难度。本次模块划分方法，根据类型，分为工具模块，功能模块，控制器模块三个大类，设计过程考虑了工具模块和功能模块的可复用性，通过控制器模块实现不同的功能。最后通过对程序的调试、相应部分功能的仿真以及在电路板的烧写，验证了整个系统的原理和本设计方案的正确性。

**关键词:** 砸彩蛋游戏机、FPGA、Verilog、Quartus

目录

[摘要 1](#_Toc59055806)

[课题设计的任务要求 3](#_Toc59055807)

[基本要求： 3](#_Toc59055808)

[提高要求： 3](#_Toc59055809)

[系统设计 4](#_Toc59055810)

[整体设计 4](#_Toc59055811)

[分块设计 5](#_Toc59055812)

[仿真波形及波形分析 10](#_Toc59055813)

[功能模块 10](#_Toc59055814)

[点阵显示模块 10](#_Toc59055815)

[数码管显示模块 11](#_Toc59055816)

[键盘输入模块 12](#_Toc59055817)

[音乐发声模块 13](#_Toc59055818)

[控制器模块 13](#_Toc59055819)

[自检模块 13](#_Toc59055820)

[主控制器模块 14](#_Toc59055821)

[功能说明及资源利用情况 15](#_Toc59055822)

[功能说明 15](#_Toc59055823)

[资源利用情况 15](#_Toc59055824)

[故障及问题分析 16](#_Toc59055825)

[总结与结论 16](#_Toc59055826)

[附 项目代码： 18](#_Toc59055827)

[Root.v 18](#_Toc59055828)

[C\_Exam.v 19](#_Toc59055829)

[C\_GameMatrix.v 20](#_Toc59055830)

[C\_Main.v 22](#_Toc59055831)

[Debounce.v 25](#_Toc59055832)

[DivideClk.v 26](#_Toc59055833)

[KeyBroad.v 27](#_Toc59055834)

[Matrix.v 28](#_Toc59055835)

[Music.v 29](#_Toc59055836)

[Num.v 30](#_Toc59055837)

[运行说明 31](#_Toc59055838)

## 课题设计的任务要求

### 基本要求：

1. 8\*8 点阵中每 2\*2 四个点为一组代表一个蛋的位置，共 4\*4个位置，与 4\*4 矩阵键

盘位置对应；

1. 数码管 DISP1 和 DISP0 显示游戏者得分，DISP7 和 DISP6 显示游戏时间；
2. 用 SW3 作为游戏机开关，打开开关 SW3 后游戏机自检：8\*8 点阵分别以红黄绿全

亮，每种颜色亮 0.5S 熄灭 0.5S；数码管 DISP7~DISP0 全亮 0.5S 熄灭 0.5S 重复三次，

进入待机状态，此时 DISP1 和 DISP0 显示得分“00”，DISP7 和 DISP6 显示游戏时

间“59”；

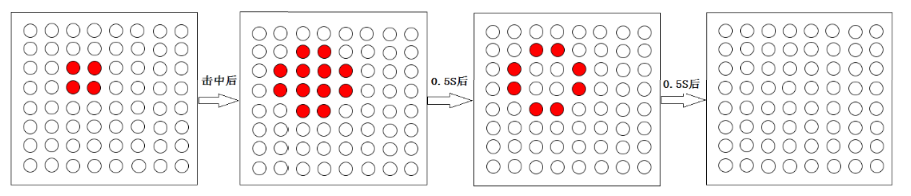
1. 使用按键 BTN3 进入游戏状态，时间从“59”开始每秒倒计时显示。点阵在随机位

置显示彩蛋，彩蛋颜色随机为红黄绿之一，每只彩蛋显示时间为 1 秒；

1. 在彩蛋显示时间内，按动相应位置的按键表示击中彩蛋，游戏者得分；彩蛋被击中

后四个亮点向四周扩散一圈以呈现爆炸效果，爆炸效果变化下图，如果彩蛋在点阵

边缘，爆炸扩散时只显示点阵范围内部分即可；



1. 击中绿蛋加 1 分，红蛋加 2 分，黄蛋加 3 分，总得分同步显示在 DISP1 和 DISP0；
2. 游戏时间未到“00”时得满 19 分则游戏结束，时间显示停止倒计时，而得分以 2Hz

闪烁显示；

1. 得分未满 19 分而游戏时间已到“00”，则游戏结束，时间显示“00”以 2Hz 闪烁，

得分稳定显示。

1. 使用按键 BTN3 可以重新进入新一轮游戏。

### 提高要求：

1. 自检过程、游戏进行时、游戏结束显示闪烁提醒时伴有适当的音乐，各个按键按下

时伴有按键音；

1. 游戏开始前，可以用 BTN0 选择彩蛋显示的时间为 2S、1S、0.5S、0.2S 之一，以改

变游戏难度；

1. 自拟其他功能。

## 系统设计

### 整体设计

设计整体按照可复用原则，将模块分成功能模块，工具模块，控制器模块三部分。功能模块为与设备相关的硬件功能抽象，包含点阵显示模块，数码管显示模块，键盘输入模块和音乐发声模块。工具模块由与硬件功能不直接相关的功能抽象而成，包含分频器模块，消抖模块和一些译码模块，这些模块在其他模块实现中使用。顶层模块为整个项目的入口，直接引用各功能模块及控制器模块，同时负责控制器与功能模块的信号连接与分配。

### 分块设计

#### 功能模块设计

功能模块是设备硬件与软件逻辑的桥梁，通过对硬件功能的抽象，给控制器逻辑更加简单易用的接口。

##### 点阵显示模块

采用128位大数据作为点阵输入，数据输入 8行 每行中8列 均按大端序排列 每格用2位表示R和G两种颜色 高电平有效。外部可忽略点阵的显示细节，仅需要传入需要显示的图案即可。

|  |  |
| --- | --- |
| 例如传入数组为  {          16'b0000000000000010,          16'b0000000000000000,          16'b0000000000000000,          16'b0000000011000000,          16'b0000001100000000,          16'b0000000000000000,          16'b0000000000000000,          16'b0100000000000000  }; | 点阵输出即（用于检验辨别方向）  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○ |

内部有一个3位计数器，控制点阵扫描。行选通信号直接使用3-8译码器输出计数器。列信号采用generate通过计算从输入data中得出结果。

module Matrix (

    input clk,*//时钟,建议1kHz*

    input [127:0] data,*//数据输入 8行 每行中8列 均按大端序排列 每格用2位表示R和G两种颜色 高电平有效*

    output [7:0] rowO,colR,colG*//输出接口*

);

    reg [2:0] row=0;*//当前显示行*

    always @(posedge clk) begin*//行计数器*

        row = row+3'd1;

    end

    D3to8 u\_D3to8(row,rowO);

    generate

        genvar i;

        for(i=0;i<8;i=i+1) begin:col

            assign colR[i] = data[(7-row)\*16+2\*i];

            assign colG[i] = data[(7-row)\*16+2\*i+1];

        end

    endgenerate

endmodule

##### 数码管显示模块

采用8位8421BCD码作为输入。内部使用一个3位计数器一个数码管译码器实现。数码管选通信号直接使用3-8译码器输出计数器值，seg通过数码管译码器输出data中是BCD码。

module Numbers (

    input clk,

    input [31:0] data,*//8\*4二进制,高位优先*

    output [7:0] segO,*//数码管管段,高电平有效*

    output [7:0] sig*//数码管扫描控制,低电平有效*

);

    reg [2:0] i = 0;

    D3to8 u\_D3to8(i,sig);

    Num u\_Num({data[4\*i+3-:4]},segO);

    always @(posedge clk ) begin

        i <= i+3'd1;

    end

endmodule

##### 键盘输入模块

将键盘转换为普通的16位按键信号。内部使用一个2位计数器实现。键盘列选通信号直接使用2-4译码器输出计数器值，并在每个时刻读入对应列键盘行信号，存入输出按键寄存器中。

如果需要消抖，可在模块外对输入按键再按照普通按键方式进行消抖。

module KeyBroad (

    input clk,rst\_n,*//时钟与重置*

    input [3:0] KB\_Row,*//键盘行列*

    output [3:0] KB\_Col,

    output reg [15:0] key *//转换后的正常按键状态,高电平有效*

);

    reg [1:0] col=0;*//键盘行计数器*

    always @(posedge clk) col = col+2'b01;*//时钟上升沿更新行*

    D2to4 u\_D2to4(col,KB\_Col);

    always @(negedge clk,negedge rst\_n) begin *//时钟下降沿读取输入*

        if(!rst\_n) key = {16{1'b0}};

        else begin

            key[col] = ~KB\_Row[3];

            key[4+col] = ~KB\_Row[2];

            key[8+col] = ~KB\_Row[1];

            key[12+col] = ~KB\_Row[0];

        end

    end

endmodule

##### 音乐发声模块

音乐发声模块使用了一个可变频分频器实现，另加一个普通分频延时器，通过音符播放完成信号。

实现了3\*7个音符，外部只需要设置好音符时长，在finish触发时，切换下一音符，即可演奏8分，4分等任意音符数量的音乐。

module Music (

    input clk,en,

    input [4:0] scale,*//音阶数 1-21,0为空音*

    input [10:0] playTime,*//持续ms数<2048*

    output finish,

    output beep

);

    parameter BaseFreq = 1\_000\_000;

*// 各音阶周期，最大为3831*

    parameter [0:32\*7\*3-1] ClkM = {

        BaseFreq/261,BaseFreq/294,BaseFreq/330,BaseFreq/349,BaseFreq/392,BaseFreq/440,BaseFreq/494,*//低音*

        BaseFreq/523,BaseFreq/587,BaseFreq/659,BaseFreq/698,BaseFreq/784,BaseFreq/880,BaseFreq/988,*//中音*

        BaseFreq/1047,BaseFreq/1175,BaseFreq/1319,BaseFreq/1397,BaseFreq/1568,BaseFreq/1760,BaseFreq/1096*//高音*

    };

    wire [11:0] M=ClkM[(scale-1)\*32+:32];

    wire clkO;

    DivideClkMN#(12) u\_DivideClkMN(clk,en,M,M>>1,clkO);

    assign beep = |scale?(en&clkO):1'b0;

    DivideClk#(BaseFreq/1000,BaseFreq/1000) u0(clk,en,clk1K);

    reg [10:0] cntT=0;

    always @(posedge clk1K or negedge en) begin

        if(!en)cntT<=0;

        else if(finish) cntT<=0;

        else cntT <= cntT+1;

    end

    assign finish = cntT>=playTime;

endmodule

#### 自检模块

#### 游戏控制模块

有一个游戏主状态机，分为4个状态：prepare （准备，等待游戏开始）gaming（游戏进行中）endTimeout（超时结束）endWin（得分获胜）

按下 主按键

计时到0

得分超过19

主按键

主按键有效

其他情况均保持状态不变

根据该状态机，控制其他功能运行

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Prepare | Gaming | End timeout | End win |
| 时间计数器 | 保持59 | 每秒递减 | 00并闪烁 | 保持时间 |
| 得分计数器 | 保持00 | 点中后加上对应得分 | 保持分数 | 保持并闪烁 |
| 音乐播放 | 播放片段1 | 打中后播放对应分数片段 | 播放音乐2 | 播放音乐3 |

##### 在Gaming状态下的具体设计

Gaming状态下，时间计数器在1hz分频时钟下正常工作。

在时钟作用下，检测键盘的按键状态是否与彩蛋位置期望位置一致，一致则增加得分，同时触发彩蛋爆炸效果（waitBoom信号），同时播放音乐。

另有一个可变延时时钟，作为彩蛋显示的超时信号。在超时信号或爆炸效果完成（finishBoom信号）后，从随机数发生器中取出下一个位置坐标及颜色。其中随机数发生器采用8位M信号发生器，为整个模块提供随机数。

多个片段声音采用从M序列发生器中特定时间取出的音符构成，因为M序列随机数为伪随机数，具有周期性，配合取音时间较快，音乐欢快富有节奏，实测效果意外好，故选择。

##### 游戏矩阵控制器

因为彩蛋游戏的点阵显示图案较为复杂，故单独抽出出游戏点阵控制器模块，接受控制器中彩蛋位置，颜色，是否爆炸的信息，输出爆炸完成的信号及点阵图案数据。

因为点阵图案需要根据参数动态计算，故将点阵data设置为reg类型，在参数变更时，data置零，再标记上需要绘制的各点颜色。

为了显示砸蛋爆炸动画，在游戏点阵控制器中，增加了一个简单摩尔型状态机，当需要显示爆炸动画时，由2Hz时钟控制，根据状态机在data上绘制颜色。在动画播放完成时，输出爆炸完成的信号。

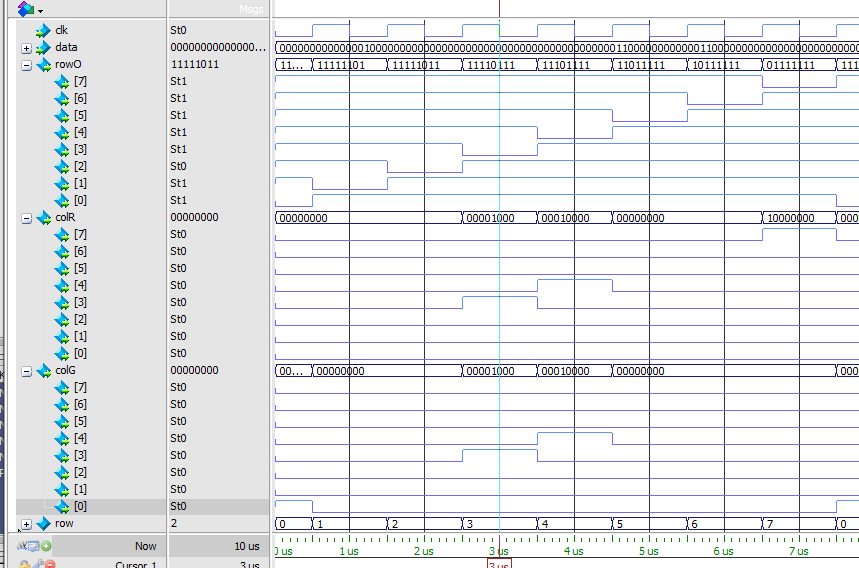
该控制器简化了游戏控制器中关于点阵显示的逻辑，使游戏的主体逻辑更加突出集中，便于开发维护。

# 仿真波形及波形分析

时钟以T=1us即1Mhz为基准

## 功能模块

### 点阵显示模块



RowO为行扫描选通信号，colR，colG为列输出信号，输入数据

data = {

        16'b0000000000000010,

        16'b0000000000000000,

        16'b0000000000000000,

        16'b0000000011000000,

        16'b0000001100000000,

        16'b0000000000000000,

        16'b0000000000000000,

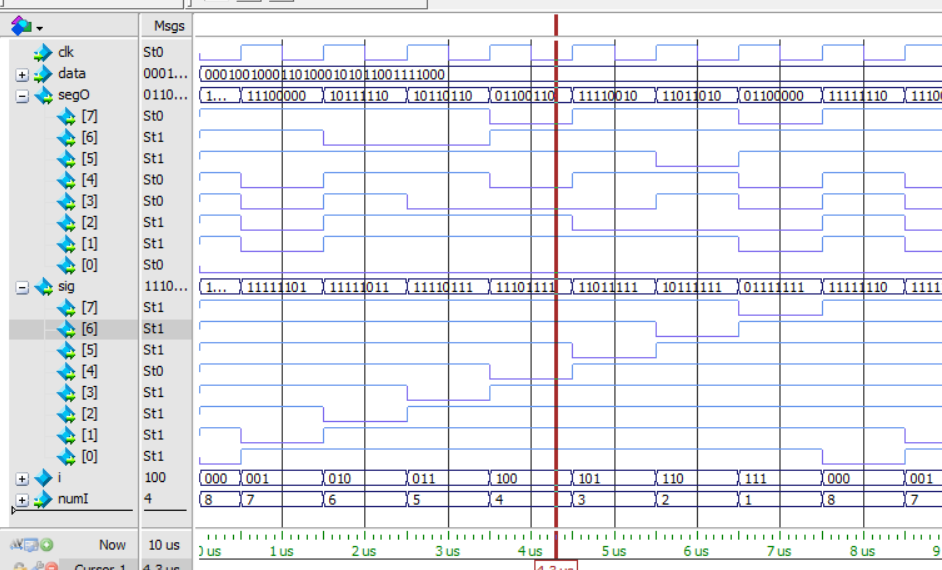
        16'b0100000000000000

};

如指针位置所示，扫描第3行（从0开数），第3列亮黄灯（红绿都亮）

也可看出，第0行0列亮绿灯，7行7列亮红灯，符合输入数据。

### 数码管显示模块



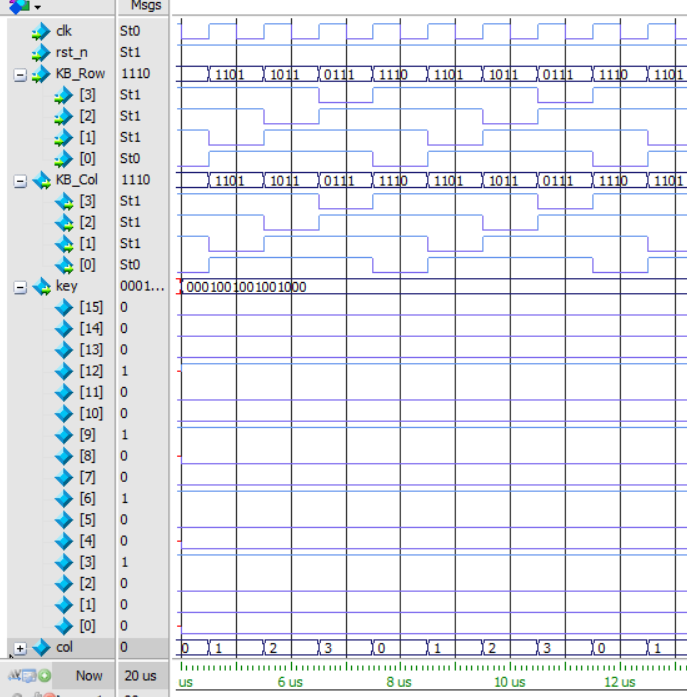
输入数据 data = ‘h12345678 图中的numI为子模块数码译码器的输入

指针处：第4号（从0开始，板子上从右到左）数码管点亮，数码管片段为bcfg，数字4

可以看出，选通信号0-7时，依次扫描显示87654321。

从左往右看，即12345678，符合输入数据。

### 键盘输入模块



仿真条件（一条斜线）

when -repeat {KB\_Col = 'b0111} {force KB\_Row 'b0111}

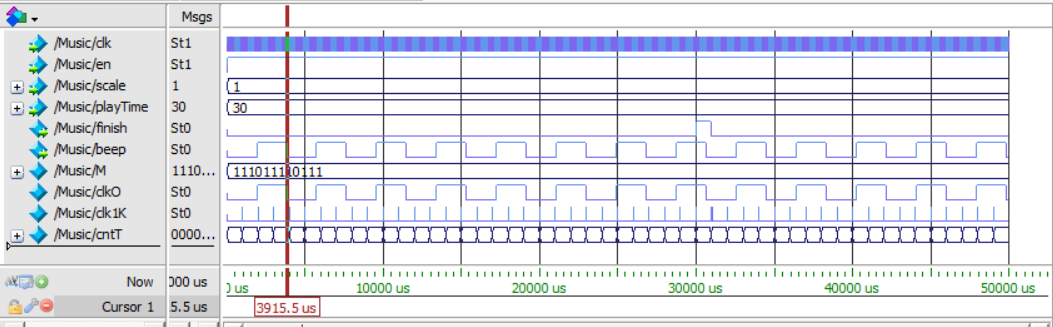
when -repeat {KB\_Col = 'b1011} {force KB\_Row 'b1011}

when -repeat {KB\_Col = 'b1101} {force KB\_Row 'b1101}

when -repeat {KB\_Col = 'b1110} {force KB\_Row 'b1110}

从图中看出，检测到3,6,9,12等按键按下，符合仿真条件

### 音乐发声模块



仿真为音符1，节拍长度30ms

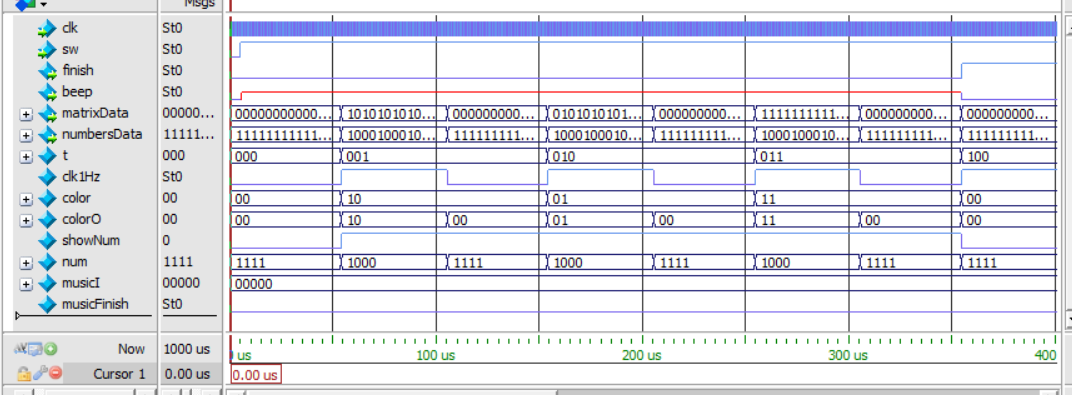
可以看到deep的周期约3.9ms，约256hz，符合C1音阶261Hz

在30ms时，finish出现高电平，提醒外层模块切换下一音符（此处外部为常量）

均符合仿真条件

## 控制器模块

### 自检模块



仿真时修改了分频器参数为100分频（正常1M分频），提高仿真速度

Beep的音乐发声模块已在上文仿真测试通过，此处因为仿真时间太短，无波形。

此处未展开matrixData和numbersData以节约空间，代码为

assign matrixData = {64{colorO}};

    assign numbersData = {8{num}};

易得matrixData即colorO重复64遍，numbersData即num重复64遍

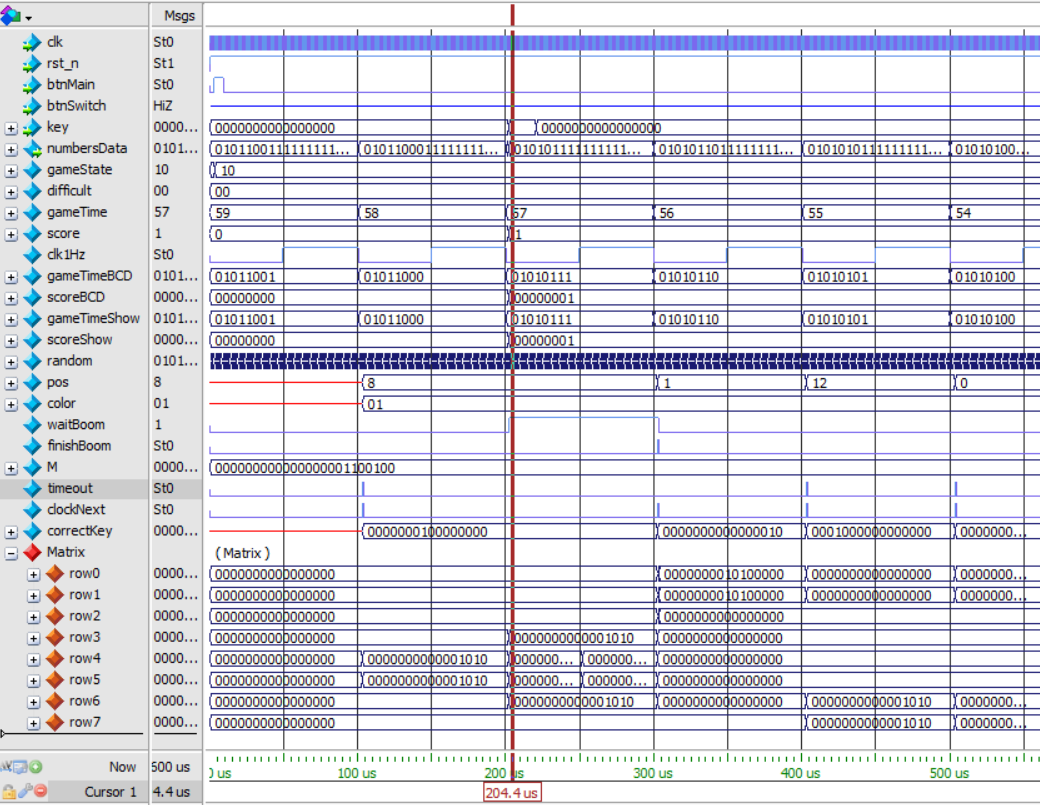
分析功能，在sw高电平时，计数器和状态机开始工作

点阵在0.5s-1s亮起红色，1.5s-2s亮起绿色，2.5s-3s亮起黄色，满足要求

数码管数据在0-3秒开始1hz,50%闪烁三次（f为不显示），满足要求。

（音频在此情况播放音频模块的21个音符，仿真中看不出，实际测试中有）

### 主控制器模块



仿真时修改了1Hz分时器参数为100分频（正常1M分频,即100us代替1s）提高仿真速度。

仿真图中删去了无法观测的音乐相关波形（音乐不便仿真）

下面从波形图中分析逻辑：

1. 开始时，btmMain触发游戏开始
2. gameTime为游戏时间，可以看到在正常倒计时，gameTimeBCD也正确工作。
3. Pos，color为彩蛋内部描述，看到每秒切换一个坐标。
4. 指针处（200us左右），仿真模拟了一个正确的按键，触发了得分+1和waitBoom信号，点阵显示也发生了变化

（仿真图的Matrix为自动整理的matrixData，同行数据左右高低位反了，不易修正）。

|  |  |  |
| --- | --- | --- |
| 按下前  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○ | 正确按下后  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○ | 0.5s后  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○  ○○○○○○○○ |

动画的总播放时间为1s，触发看finishBoom信号，满足要求。

1. 动画播放完成后，下一个坐标1（从0开始），矩阵显示满足要求（考虑仿真显示相反）
2. 音乐功能已通过上机测试，确认正常。

# 功能说明及资源利用情况

## 功能说明

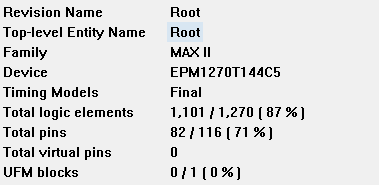
以时间顺序

1. 开机后，默认休眠状态，若开启SW3（或已开启），则开始自检。
2. 自检时，点阵依次以红绿黄闪烁，1Hz，50%；数码管闪烁3次，音乐模块以1/8播放21个音符。然后交接控制权给游戏主模块
3. 进入游戏模块后首先为等待状态，时间显示59，默认难度1，得分0，播放背景音乐1。
   1. 该状态下可以使用BTN0修改难度，有0123共4个档，分别代表（2s，1s，0.5s，0.2s显示时间）
   2. 按下BTN3开始游戏。
4. 开始游戏后，会在随机位置显示随机颜色的彩蛋，彩蛋显示时间由难度决定。
   1. 若此时按下正确的按键，则会显示爆炸动画，得分加上颜色对应得分，同时有一段简单的音效。
   2. 若没按下正确按键，则在显示结束后切换刷新下一个彩蛋位置。
   3. 如果得分超过19分，进入获胜状态；如果时间结束，进入超时结束状态。
5. 获胜状态下，播放音乐2，得分闪烁，时间保持不变。

超时结束状态，播放音乐3，得分不变，时间闪烁。

1. 再次按下BTN3，恢复到游戏等待状态，回到流程3.。

## 资源利用情况



关于使用管脚：在底层模块中将LED，开关，按钮等全部声明，但并未全部使用，详细使用情况见下方。

板载资源使用：1Mhz时钟，8\*8点阵，键盘按键，数码管（自检全部使用，游戏只使用6个），LED灯（使用4个指示内部状态，可去除），开关（实际只使用SW3），按钮（实际使用BTN3和BTN0），蜂鸣器。

# 故障及问题分析

1. 检测自检模块前，发现开关没反应：

**解：**后发现板载时钟设置不正确，设置后问题解决。以后实测前均先检测时钟设置

1. 编译无错误，运行逻辑混乱，后检测编译发现严重警告

Critical Warning (10237): Verilog HDL warning at C\_Main.v(65): can't infer register for assignment in edge-triggered always construct because the clock isn't obvious. Generated combinational logic instead

**解：**原来是always@(posedge timeout1s or posedge finishBoom)

编译器无法编译出正确的触发沿，后改为

    wire clockNext = timeout | finishBoom;*//解决10237问题*

    always@(posedge clockNext)begin

问题解决

1. 游戏状态机受score等数值变量冒险影响

**解：**

* 1. 更改状态机本身状态编码,尽量减少状态本身冒险
  2. 增加clk1khz分频时钟控制状态机,减少外部状态导致冒险的几率

1. 游戏点阵控制器，爆炸效果溢出到不正确的行

**解：**在爆炸绘制的地方，增加if条件，避免边界溢出错误显示问题。

1. 本想使用row<<5+col<<2优化性能,但移位会丢失位数

    wire [1:0] row,col;

    assign row = pos[3:2];

    assign col = pos[1:0];、

    wire [6:0] base = row<<5+col<<2;

**解：**后改为，不影响效果，而且更加优雅

wire [6:0] base = {row,1'h0,col,2'h0};

1. 编译时，提示逻辑单元数量不足

**解：**

* 1. 升级分频器模块以支持变频, 改变之前使用多个分频器来支持变频的方式，大大减少逻辑单元数量
  2. 使用类似x\*103>>10代替%10的技巧，减少算数逻辑对逻辑单元的消耗。

# 总结与结论

一开始选择这个课题的时候，并没有在乎它的难度，而是觉得这个课题如果做出来最有实际性的成就感。不像郁金香那种无聊的养成游戏，只有几个简单的动画，也不像LED点阵风扇，光看到转动却感觉不到一点凉风。我不喜欢虚的，我喜欢实的东西，能切实带来用处的东西。彩蛋机，有动画，有声音，更有实际的交互和玩耍过程，完成的不止是冰冷冷的一大堆代码，而是一个完成之后立马就可以带来乐趣的游戏机。

按照老师们精心设计的各个实验题目中，彩蛋机排在最后，是唯一一个三星级难度的实验，在第一节课时，老师也说这个题目确实有难度。但是有难度更有动力，尽管在实验设计中遇到了各种各样的问题麻烦，但我也并没有气馁，反而越战越勇，通过问题提高自己解决问题的能力。最终功夫不负有心人，成功完成了实验，体验着自己制作的彩蛋机，这其中的心情无法用语言形容，但相信每一个成功做出实验的人都能体会到。

Verilog从数电零的开始，从简单的点亮LED，到按键计数器，到最终的彩蛋机，难度越来越高。这其中越来越感觉到Verilog基础语法的重要性，在一次又一次修改自己的代码的过程中，明白了别人示例在本来不明白写法的原因，并不断改进自己的代码，改进自己的项目结构，设计出属于自己的彩蛋机。实验也大大加深了自己对于理论课的理解，记得在有一次上数电课时，老师介绍了M数列发生器，当时还不知道这种数列到底有什么作用，直到这次，我在自己的彩蛋机中使用了M数列发生器作为随机数发生器，并且还意外发现了M数列随机音乐。

在最后，非常感谢数电课和数电实验课老师们，感谢老师们精心设计的教学计划，还有老师们的精心的启发教学与指导。没有这些，我的实验过程肯定会更加艰辛！老师，您们辛苦了！

# 附 项目代码：

## Root.v

module Root (*//此处定义硬件相关引脚信息,不一定全部使用*

    input clk,*//时钟取1Mhz*

    input [7:0] sw,*//上为高电平*

    input [7:0] btn,*//按键,高电平有效*

    output [15:0] ledO,*//LED输出端口,高电平有效*

    output beep,*//蜂鸣器输出*

    input [3:0] KB\_Row,*//键盘行列,高电平有效*

    output [3:0] KB\_Col, *//低电平有效*

    output [7:0] rowO,colR,colG,*//点阵输出接口*

    output [7:0] segO,sig*//数码管输出控制接口*

);

    wire btnMain,btnSwitch;

    Debounce#(2) u\_Debounce(clk,{btn[3],btn[0]},{btnMain,btnSwitch});

    wire swMain = sw[3];

    wire rst\_n = 1'b1;

*// 信号*

    wire finishExam,enableMain;

    assign enableMain = sw[7]|(swMain&finishExam);*//使用sw7可以强行跳过自检*

    wire [127:0] matrixData,matrixDataExam,matrixDataMain;

    assign matrixData = enableMain?matrixDataMain:matrixDataExam;

    wire [31:0] numbersData,numbersDataExam,numbersDataMain;

    assign numbersData = enableMain?numbersDataMain:numbersDataExam;

    wire beepExam,beepMain;

    assign beep = enableMain?beepMain:beepExam;

    wire [15:0] key;

*//控制模块*

    Exam u\_Exam(clk,swMain,finishExam,beepExam,matrixDataExam,numbersDataExam);

    Main u\_Main(clk,enableMain,btnMain,btnSwitch,key,beepMain,matrixDataMain,numbersDataMain);

*//功能模块*

    Matrix u\_Matrix(clk,matrixData,rowO,colR,colG);

    Numbers u\_Numbers(clk,numbersData,segO,sig);

    KeyBroad u\_KeyBroad(clk,rst\_n,KB\_Row,KB\_Col,key);

    assign ledO[0]=finishExam;

    assign ledO[1]=enableMain;

    assign ledO[2]=btnMain;

    assign ledO[3]=btnSwitch;

endmodule

## C\_Exam.v

module Exam (

    input clk,sw,*//sw为主开关状态*

    output finish,*//完成自检状态指示,高电平有效*

    output beep,*//蜂鸣器输出*

    output [127:0] matrixData,*//点阵模块数据*

    output [31:0] numbersData*//数码管模块数据*

);

    reg [2:0] t = 0;*//状态,从自检开始的秒数*

    wire clk1Hz;*//分频时钟,1Hz 50%*

    DivideClk u\_DivideClk(clk,sw,clk1Hz);*//分频器*

    assign finish = (t==4);*//t=7时标志着自检完成*

    reg [1:0] color;*//矩阵颜色RG*

    wire [1:0] colorO = color &{clk1Hz,clk1Hz};*//通过1Hz方波掩码后的颜色*

    assign matrixData = {64{colorO}};

    reg showNum=0;*//是否显示数码管*

    wire [3:0] num = (showNum&clk1Hz)?4'h8:4'hf;*//通过方波控制显示数字为8(全亮)或f(隐藏)*

*// assign numbersData[31:28] = {1'b0,t};//显示自检状态*

    assign numbersData[31:0] = {8{num}};

    reg [4:0] musicI=0;

    wire musicFinish;

    Music u\_Music(clk,sw&~finish,musicI,125,musicFinish,beep);

    always @(posedge musicFinish or negedge sw) begin

        if(!sw) musicI <= 1'b0;

        else if (musicI == 5'd21)*//播放完最后一个音符*

            musicI <= 1'b0;

        else

            musicI <= musicI+1'b1;

    end

    always @(posedge clk1Hz or negedge sw) begin

        if(!sw) begin

            t<=3'b0;

            color <=2'h0;

            showNum=0;

        end

        else if(!finish) begin

            t <= t+3'h1;

            case (t)

                0:  begin

                    color <= 2'b10;*//红*

                    showNum <= 1'b1;*//闪烁数字*

                end

                1:  color <= 2'b01;*//绿*

                2:  color <= 2'b11;*//黄*

                3: begin

                    color <= 2'b00;

                    showNum <= 1'b0;

                end

                default: t<=0;*//不应当出现*

            endcase

        end

    end

endmodule

## C\_GameMatrix.v

*//游戏点阵控制器*

module GameMatrixDisplay (

    input clk,en,

    input [3:0] pos,

    input [1:0] color,

    input showBoom,*//显示爆炸动画*

    output reg finishBoom,*//爆炸动画完成*

    output reg [127:0] matrixData

);

    reg boomState;*//0为第一动画,1为第二动画*

    wire clk2Hz;*//分频时钟,2Hz脉冲,0.5s*

    DivideClk#(.M(500\_000)) u\_DivideClk(clk,showBoom,clk2Hz);*//分频器*

    always @(negedge showBoom or posedge clk2Hz) begin

        if(!showBoom)begin

            boomState <=1'd0;

            finishBoom <= 1'b0;

        end else begin

            if(boomState) finishBoom <= 1'b1;

            else boomState <= 1'd1;

        end

    end

    wire [1:0] row,col;

    assign row = pos[3:2];

    assign col = pos[1:0];

    wire [6:0] base = {row,1'h0,col,2'h0};

    always @(\*) begin

        matrixData = 128'h0;*//清屏*

        if(en) begin

            if (~boomState) begin*// 0状态时,显示中间4点 showBoom为任意值，无需判断*

*//         base 配上偏移值*

                matrixData[base+0]=color[1];

                matrixData[base+1]=color[0];

                matrixData[base+2]=color[1];

                matrixData[base+3]=color[0];

                matrixData[base+16]=color[1];

                matrixData[base+16+1]=color[0];

                matrixData[base+16+2]=color[1];

                matrixData[base+16+3]=color[0];

            end

            if(showBoom)begin*//  0|1均时,显示周围8点 boomState为任意值，无需判断*

                if(|row)begin*//防止上溢出*

                    matrixData[base-16+0]=color[1];

                    matrixData[base-16+1]=color[0];

                    matrixData[base-16+2]=color[1];

                    matrixData[base-16+3]=color[0];

                end

                if(|col)begin*//防止左溢出*

                    matrixData[base-2]=color[1];

                    matrixData[base-1]=color[0];

                    matrixData[base+16-2]=color[1];

                    matrixData[base+16-1]=color[0];

                end

                if (~&col) begin*//防止右溢出*

                    matrixData[base+4]=color[1];

                    matrixData[base+5]=color[0];

                    matrixData[base+16+4]=color[1];

                    matrixData[base+16+5]=color[0];

                end

                if (~&row) begin*//防止下溢出*

                    matrixData[base+32+0]=color[1];

                    matrixData[base+32+1]=color[0];

                    matrixData[base+32+2]=color[1];

                    matrixData[base+32+3]=color[0];

                end

            end

        end

    end

endmodule

## C\_Main.v

module Main (

    input clk,rst\_n,

    input btnMain,btnSwitch,

    input [15:0] key,*//键盘按键*

    output beep,

    output [127:0] matrixData,

    output [31:0] numbersData

);

    localparam Perpare = 2'h1;

    localparam Gaming = 2'h2;

    localparam EndTimeOut = 2'h0;

    localparam EndWin = 2'h3;

    reg [1:0] gameState=Perpare;

    reg [1:0] difficult=0;*//难度，数字越大，反应时间越少*

    reg [7:0] gameTime,score;

    wire clk1Hz;*//分频时钟,1Hz 50%*

    DivideClk u\_DivideClk(clk,1'b1,clk1Hz);*//分频器*

*//游戏状态机时钟,分频降低速度,避免冒险*

    wire clk1k;*//分频时钟,1k Hz 50%*

    DivideClk#(.M(1\_000),.N(500)) u2\_DivideClk(clk,1'b1,clk1k);

*//主游戏状态机*

    always @(posedge clk1k) begin

        if(!rst\_n)gameState<=Perpare;

        else begin

            gameState <= gameState;

            case (gameState)

                Perpare: if(btnMain&&gameTime==8'd59)gameState<=Gaming;*//增加时间判断,确保各项参数初始化完成,游戏状态稳定*

                Gaming: begin

                    if(gameTime==0)

                        gameState<=EndTimeOut;

                    else if(score>=19)

                        gameState<=EndWin;

                end

                EndTimeOut,EndWin: if(btnMain)gameState<=Perpare;

            endcase

        end

    end

*//数码管显示控制区*

    wire [7:0] gameTimeBCD,scoreBCD;*//BCD表示数*

    D8421toBCD d1(gameTime,gameTimeBCD),d2(score,scoreBCD);

    wire [7:0] gameTimeShow = (gameState==EndTimeOut & clk1Hz)?8'hff:gameTimeBCD;

    wire [7:0] scoreShow = (gameState==EndWin & clk1Hz)?8'hff:scoreBCD;

    assign numbersData = {gameTimeShow,8'hff,2'h0,difficult,4'hf,scoreShow};

*//游戏时间计数器*

    always @(negedge clk1Hz)begin

        case (gameState)

            Perpare: gameTime<=8'd59;

            Gaming: gameTime<=gameTime-1'd1;

            EndTimeOut: gameTime<=1'd0;

            EndWin: gameTime<=gameTime;

        endcase

    end

*//M序列发生器产生随机数*

    reg [7:0] random=8'b1;

    always @(posedge clk) begin

        if(!rst\_n)random=8'b1;

        else random = {random[6:0],random[7]^random[0]};

    end

*//难度控制*

    always @(posedge btnSwitch) begin

        if(gameState!=Gaming)*//游戏中不能切换*

            difficult = difficult+1;

    end

*//游戏主体逻辑处理*

    reg [3:0] pos;

    reg [1:0] color;*//颜色RG,同时代表分数*

    reg waitBoom;*//中间间隔*

    wire finishBoom;

    localparam [0:95] Ms = {24'd2000\_000,24'd1000\_000,24'd500\_000,24'd200\_000};

    wire [23:0] M= Ms[difficult\*24+:24];

    wire timeout;*//延时时钟*

    DivideClkMN#(24) u\_DivideClkMN(clk,(gameState==Gaming)&&(~waitBoom),M,M,timeout);*//延时器*

    GameMatrixDisplay u\_GameMatrixDisplay(clk,gameState==Gaming,pos,color,waitBoom,finishBoom,matrixData);*//游戏点阵控制器*

*//获取下一个坐标点*

    wire clockNext = timeout | finishBoom;*//解决10237问题*

    always@(posedge clockNext)begin

        pos <= random[7:4];

        color[1] <= random[1];*//取随机数*

        color[0] <= ~random[1] | random[0]; *//保证不均为0*

    end

*//按键检测及得分控制*

    wire [15:0] correctKey = (16'b1)<<pos;

    always @(posedge clk) begin

        if(gameState==Perpare)score <=0;

        if(gameState==Gaming)begin

            if(finishBoom) waitBoom = 1'b0;

            else if(!waitBoom)begin

                if(key==correctKey)begin*//当前仅按下正确的按键*

                    waitBoom = 1'b1;

                    score <= score+color;

                end

            end

        end else waitBoom = 1'b0;

    end

*//======声音控制======*

    reg [4:0] musicData;

    wire finishMusic;

    Music u\_Music(clk,rst\_n,timeout?(pos+1'b1):(musicData),180,finishMusic,beep);

    always @(posedge clk) begin

        if(clockNext)begin

            musicData <= pos+1'b1;*//插入一个音符的提示音*

        end else if (btnMain) begin

            musicData <= 5'h6;*//主按键提示音*

        end else if(finishMusic)begin

            case(gameState)*//基于伪随机数产生简单音频*

                Perpare: musicData <= {random[6]^random[3],random[3],random[0]|random[1],1'b1};

*// Perpare: musicData <= random[2:0]+5'b1;*

                Gaming: musicData <= waitBoom?((color-1)\*7+{random[2],random[0],1'b1}):5'b0;

                EndWin: musicData <= {random[6],random[3],random[4],random[0]^random[4]};

                EndTimeOut: musicData <= random[2:0]+random[5:3];

                default: musicData <= 0;

            endcase

        end

    end

endmodule

module D8421toBCD (

    input [7:0] I,

    output [7:0] O

);

*//使用乘法伪除法，降低资源消耗（确保60以下无误）*

    assign O[7:4] = (I\*13'd103)>>10;

    assign O[3:0] = I % 4'd10;

endmodule

## Debounce.v

*//消抖模块(20ms)*

module Debounce(clk,I,O);

    parameter Size = 1;

    parameter ClkSpeed = 1\_000\_000;*//传入时钟速度*

    input clk;

    input [Size-1:0] I;

    output reg [Size-1:0] O;

    reg [Size-1:0] pre,now;

    always @ (posedge clk) begin

        pre <=now;

        now <= I;

    end

    wire clkEnable = ~|(pre^now);

    wire clkO;

    DivideClk #(.M(ClkSpeed/50),.N(ClkSpeed/52)) u\_DivideClk(clk,clkEnable,clkO);*//20ms延时器*

    initial O=I;

    always @(posedge clkO) O=pre;

endmodule

## DivideClk.v

*/\*\**

*\* 分时器模块*

*\* enable控制分时器的运行*

*\* 通过M,N调节循环周期和占空比*

*\*/*

module DivideClk(

    input clkI,

    input enable,*//0时关闭计算器,且归零*

    output reg clkO

);

*//  parameter WIDTH = 24;*

    parameter M = 1\_000\_000;

    parameter N = 500\_000;

    localparam WIDTH = $clog2(M+1);

    reg [WIDTH-1:0] r=1;*//在1->M中循环*

    always @ (posedge clkI or negedge enable) begin

        if(!enable) begin*//未启用,全部置零*

            clkO <= 1'b0;

            r <= 1'b1;

        end else begin

            clkO <= r>=N || r==M;

            if(r==M)

                r<=1'b1;

            else

                r<=r+1'b1;

        end

    end

endmodule

*//可变M，N的分时器*

module DivideClkMN#(parameter WIDTH = 24)(

    input clkI,

    input enable,*//0时关闭计算器,且归零*

    input [WIDTH-1:0] M,N,

    output reg clkO

);

    reg [WIDTH-1:0] r=1;*//在1->M中循环*

    always @ (posedge clkI or negedge enable) begin

        if(!enable) begin*//未启用,全部置零*

            clkO <= 1'b0;

            r <= 1'b1;

        end else begin

            clkO <= r>=N || r==M;

            if(r>=M)

                r<=1'b1;

            else

                r<=r+1'b1;

        end

    end

endmodule

## KeyBroad.v

module KeyBroad (

    input clk,rst\_n,*//时钟与重置*

    input [3:0] KB\_Row,*//键盘行列*

    output [3:0] KB\_Col,

    output reg [15:0] key *//转换后的正常按键状态,高电平有效*

);

    reg [1:0] col=0;*//键盘行计数器*

    always @(posedge clk) col = col+2'b01;*//时钟上升沿更新行*

    D2to4 u\_D2to4(col,KB\_Col);

    always @(negedge clk,negedge rst\_n) begin *//时钟下降沿读取输入*

        if(!rst\_n) key = {16{1'b0}};

        else begin

            key[col] = ~KB\_Row[3];

            key[4+col] = ~KB\_Row[2];

            key[8+col] = ~KB\_Row[1];

            key[12+col] = ~KB\_Row[0];

        end

    end

endmodule

module D2to4 (

    input [1:0] data,

    output reg [3:0] O*//输出,高电平有效*

);

    always @(data) begin

        case (data)

            3'h0: O=4'b1110;

            3'h1: O=4'b1101;

            3'h2: O=4'b1011;

            3'h3: O=4'b0111;

            default: O=4'b1111;

        endcase

    end

endmodule

## Matrix.v

module Matrix (

    input clk,*//时钟,建议1kHz*

    input [127:0] data,*//数据输入 8行 每行中8列 均按大端序排列 每格用2位表示R和G两种颜色 高电平有效*

    output [7:0] rowO,colR,colG*//输出接口*

);

    reg [2:0] row=0;*//当前显示行*

    always @(posedge clk) begin*//行计数器*

        row = row+3'd1;

    end

    D3to8 u\_D3to8(row,rowO);

    generate

        genvar i;

        for(i=0;i<8;i=i+1) begin:col

            assign colR[i] = data[(7-row)\*16+2\*i];

            assign colG[i] = data[(7-row)\*16+2\*i+1];

        end

    endgenerate

endmodule

module D3to8 (

    input [2:0] data,

    output reg [7:0] O*//输出,低电平有效*

);

    always @(data) begin

        case (data)

            3'h0: O=8'b1111\_1110;

            3'h1: O=8'b1111\_1101;

            3'h2: O=8'b1111\_1011;

            3'h3: O=8'b1111\_0111;

            3'h4: O=8'b1110\_1111;

            3'h5: O=8'b1101\_1111;

            3'h6: O=8'b1011\_1111;

            3'h7: O=8'b0111\_1111;

            default: O=8'b1111\_1111;

        endcase

    end

endmodule

## Music.v

module Music (

    input clk,en,

    input [4:0] scale,*//音阶数 1-21,0为空音*

    input [10:0] playTime,*//持续ms数<2048*

    output finish,

    output beep

);

    parameter BaseFreq = 1\_000\_000;

*// 各音阶周期，最大为3831*

    parameter [0:32\*7\*3-1] ClkM = {

        BaseFreq/261,BaseFreq/294,BaseFreq/330,BaseFreq/349,BaseFreq/392,BaseFreq/440,BaseFreq/494,*//低音*

        BaseFreq/523,BaseFreq/587,BaseFreq/659,BaseFreq/698,BaseFreq/784,BaseFreq/880,BaseFreq/988,*//中音*

        BaseFreq/1047,BaseFreq/1175,BaseFreq/1319,BaseFreq/1397,BaseFreq/1568,BaseFreq/1760,BaseFreq/1096*//高音*

    };

    wire [11:0] M=ClkM[(scale-1)\*32+:32];

    wire clkO;

    DivideClkMN#(12) u\_DivideClkMN(clk,en,M,M>>1,clkO);

    assign beep = |scale?(en&clkO):1'b0;

    DivideClk#(BaseFreq/1000,BaseFreq/1000) u0(clk,en,clk1K);

    reg [10:0] cntT=0;

    always @(posedge clk1K or negedge en) begin

        if(!en)cntT<=0;

        else if(finish) cntT<=0;

        else cntT <= cntT+1;

    end

    assign finish = cntT>=playTime;

endmodule

## Num.v

module Num(

    input [3:0] numI,

    output reg [7:0] segO

);

    always@(numI) case(numI)

        4'h0: segO <= 8'b1111\_1100;

        4'h1: segO <= 8'b0110\_0000;

        4'h2: segO <= 8'b1101\_1010;

        4'h3: segO <= 8'b1111\_0010;

        4'h4: segO <= 8'b0110\_0110;

        4'h5: segO <= 8'b1011\_0110;

        4'h6: segO <= 8'b1011\_1110;

        4'h7: segO <= 8'b1110\_0000;

        4'h8: segO <= 8'b1111\_1110;

        4'h9: segO <= 8'b1111\_0110;

*// 4'ha: segO <= 8'b1110\_1110;*

*// 4'hb: segO <= 8'b0011\_1110;*

*// 4'hc: segO <= 8'b1001\_1100;*

*// 4'hd: segO <= 8'b0111\_1010;*

*// 4'he: segO <= 8'b1001\_1110;*

*// 4'hf: segO <= 8'b1000\_1110;*

        default: segO= 8'h00;

    endcase

endmodule

module Numbers (

    input clk,

    input [31:0] data,*//8\*4二进制,高位优先*

    output [7:0] segO,*//数码管管段,高电平有效*

    output [7:0] sig*//数码管扫描控制,低电平有效*

);

    reg [2:0] i = 0;

    D3to8 u\_D3to8(i,sig);

    Num u\_Num({data[4\*i+3-:4]},segO);

    always @(posedge clk ) begin

        i <= i+3'd1;

    end

endmodule

## 运行说明

为了方便运行调试，将代码打包附在最后

另外，本项目开源放在<https://github.com/way-zer/FPGA>（觉得好可以给个Star）

|  |  |
| --- | --- |
| 打包文件，可导出解压打开 | C:\Users\q1048\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\CA1BB691.tmp  开源地址 二维码 |