# 北京邮电大学电路实验中心
# <数字系统设计实验>

# 实
# 验
# 报
# 告

实验名称： 打地鼠游戏的设计与实现

班　　级： 2017211128 姓名（班内序号）：罗浩(1)

学　　院：信息与通信工程学院 学　　号： 2017210017

实 验 室： 电路中心实验室（三） 实验组别： 1

实验时间： 2018 年 12 月

审阅教师： 评定成绩：

# 目录

# 一、 设计课题的任务要求

## 1. 基本要求：

（1） 设计一个挑战反应速度的“打地鼠”游戏， 采用 8×8 双色点阵显示游戏界面，其中游戏边界采用绿色 LED 显示，随机出现的地鼠采用红色 LED 显示， 游戏有 16 个洞穴，如下图所示：

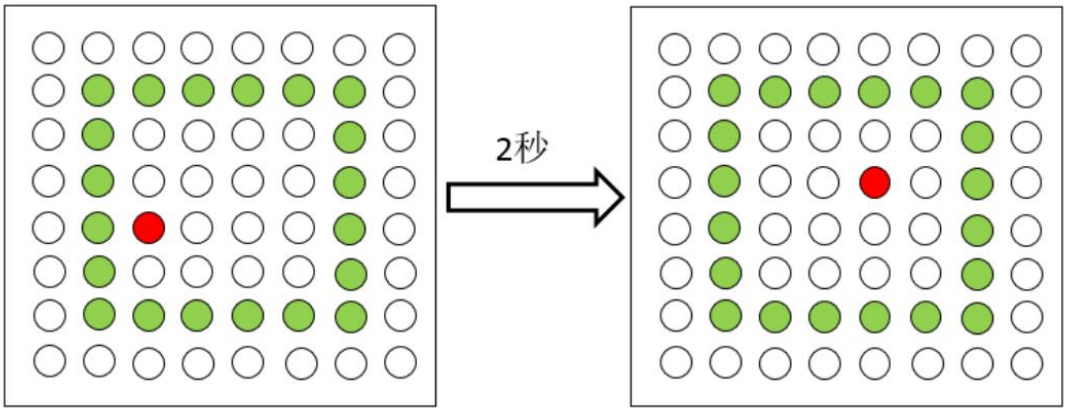


图 1 实验基本要求示意图

（2） 游戏洞穴每次随机出现一个地鼠，每个地鼠的最长保持时间为 2 秒，2 秒后随机出现下一个地鼠。以 4×4 键盘的按键代表锤子， 16 个洞穴与 16 个按键一一对应，一旦锤子在 2 秒内击中地鼠，地鼠消失，数码管计分器分数加 1 分；若锤子一直没有击中地鼠， 2 秒后该地鼠消失。用两个数码管显示游戏成绩，当游戏成绩达到 10 分时游戏结束，点阵显示绿色笑脸；

（3） 用两个数码管对整个游戏进行倒计时， 当游戏时间超过 39 秒而成绩未达到 10 分时，游戏失败，点阵显示红色哭脸；

（4） 按复位键重新开始游戏，并开始倒计时。

## 2. 提高要求：

（1） 增加游戏难度，在边界内每次随机出现两个地鼠，两个地鼠的最长保持时间均为 2 秒， 2 秒后随机出现下两个地鼠，锤子击中一个地鼠加 1 分，当游戏成绩达到 20 分而且游戏时间未超过 59 秒时，游戏结束，显示绿色笑脸，否则显示红色哭脸；

（2） 自拟其他功能。

# 二、 系统设计

## 1. 设计思路

本系统的设计采用自顶向下的设计原则。

首先分析系统的工作循环。结合设计要求，我将系统的工作循环划分为四个阶段，分别对应四种状态：初始状态、游戏状态、游戏成功状态、游戏失败状态。因此需要设计一个模块控制系统在这四种状态中转移，即有限状态机。

初始状态：当复位键被按下时，整个系统进入初始状态。此时计分数码管显示分数的初始值 00 分，倒计时数码管显示游戏时间初始值 40 秒，点阵显示绿色边框。

游戏状态：松开复位键后，系统立刻进入游戏状态。此时点阵显示绿色边框和一只随机出现的红色地鼠。当锤子打中地鼠之后地鼠立刻消失，直到新的地鼠产生，同时计分加 1，蜂鸣器发声；如果未击中则在两秒后出现一只新的地鼠。如果分数达到 10 分且游戏时间未到，系统转入游戏成功状态；如果分数不够 10 分且游戏时间已到，系统转入游戏失败状态；否则，系统将保持游戏状态。

游戏成功状态：点阵显示绿色笑脸，计分数码管显示游戏得分。倒计时停止，倒计时数码管显示剩余时间。按下复位键后转入初始状态，否则系统保持在当前状态。

游戏失败状态：点阵显示红色哭脸，计分数码管显示游戏得分，倒计时数码管显示剩余时间 00s，按下复位键后系统进入初始状态，否则系统保持在当前状态。

根据以上分析得到系统的逻辑流程图如图 2 所示。

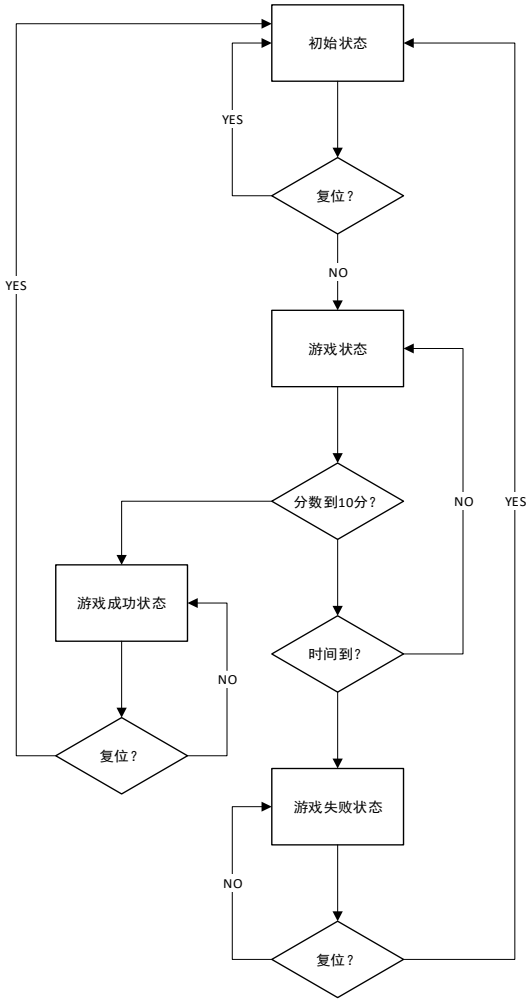


图 2 系统的逻辑流程图

系统的状态转移图如图 3 所示。图 3 中 S0 代表初始状态，S1 代表游戏状态，S2 代表游戏成功状态，S3 代表游戏失败状态；RST 代表复位信号，高电平有效；Ts=1 代表分数达到 10 分，Tt=1 代表时间到；Ts’代表 Ts 的非，Ts’Tt 代表 Ts’和 Tt 的逻辑乘。

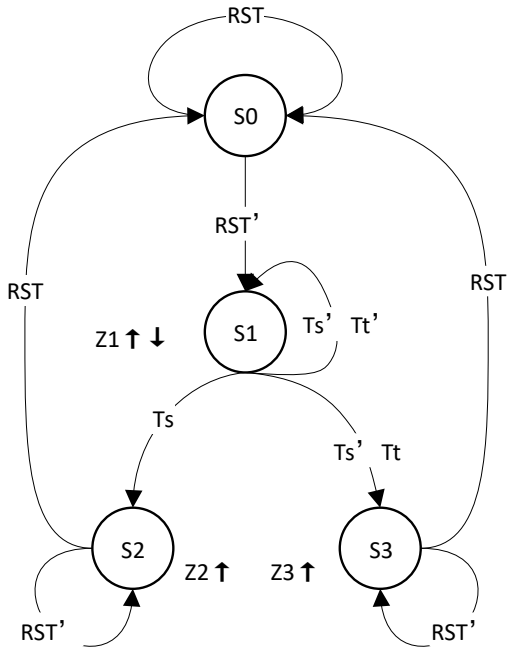


图 3 系统的状态转移图

电路的判断、控制均由控制模块完成。控制模块读取计时模块的时间产生 Tt 信号，进行计分并产生 Ts 信号，这两个信号传入有限状态机控制状态机的状态转移。

在游戏状态 S1 中地鼠需要随机产生，所以系统需要设计随机数生成模块。

由于系统各部分使用不同的扫描或工作频率，因此在系统中还需要设计分频模块。系统需要倒计时模块，这个模块的倒计时是否进行受到控制模块的控制。系统需要将分数、时间的 BCD 码转换成控制七段数码管各段亮灭的信号，因此系统需要设计译码模块。此外系统还需要控制点阵的显示、数码管的显示、蜂鸣器发声，以及读取键盘的输入，对应地分别需要设计点阵显示模块、数码管位选扫描模块、蜂鸣器驱动模块、键盘扫描模块。

# 2. 分块设计

（1） 有限状态机 fsm

有限状态机模块的设计采用三段式状态机的一般设计思想，用两个 always 块分别描述状态寄存器和次态逻辑。由于状态机的输出就是状态 state 本身，所以没有另用一个 always 块描述输出逻辑。fsm 输出的 state 控制系统中所有模块的显示和工作。输入 Ts 和 Tt 控制状态机的状态转移。

（2） 控制模块 controller

这个模块负责电路的判断功能，同时产生“打中”的信号 hit。键盘扫描模块输出的按下的按键的行列坐标 row 和 col 传入这个模块。同时随机数生成模块生成的随机数 rowran 和 colran 也传入这个模块，（rowran，colran）就是点阵上地鼠显示的位置坐标。当键盘按键的坐标和地鼠位置的坐标对应时，输出“打中”信号 hit=1，同时分数加 1。模块中用一个 always 块检测分数的变化，当分数达到 10 分（也就是 score_shi==1）时 Ts 变为 1，传入 fsm 控制 state 变为 S2（游戏成功状

态）。同时用一个 always 块检测倒计时时间的变化，一旦时间到 Tt 就变为 1，传入 fsm 控制 state 变为 S3（游戏失败状态）。

（3） 随机数生成模块 getrandom

这个模块基于 M 序列发生器生成伪随机数，使用 6 位 D 触发器级联组成的移位寄存器。从 6 个 D 触发器的 6 个输出端取出两个 3 位二进制随机数，高三位作为第 1 个随机数，低三位作为第 2 个随机数。再将高三位对 4 取模的结果再加 2 作为地鼠的行坐标 rowran，低三位同样处理后的结果作为地鼠的列坐标 colran，这样可以保证产生的地鼠位于点阵中央 4×4 的区域内。这个模块接 0.5Hz 时钟，两秒触发一次移存器移位，产生新地鼠。

（4） 分频模块 time_divider

因为这个模块在系统中要复用多次，所以我把分频比和计数器位宽设计成参数，方便在顶层模块中调用，同时也便于节约器件资源。偶分频的基本原理：如果分频后时钟 clk_out 的频率是激励时钟 clk 频率的$\frac{1}{N}$（N 为偶数），那么 clk_out 需要在计数器 cnt 计到$\frac{N}{2}-1$时翻转，cnt 重新从 0 开始计数；当 cnt<$\frac{N}{2}-1$时，在 clk 的每一个有效边沿 cnt 都要加 1，此时 clk_out 保持不变。模块就是基于这个原理设计的。

（5） 键盘扫描模块 keyboard_scan

矩阵键盘扫描的实现原理是可编程器件向矩阵键盘的列信号依次写‘0’，同时读取行信号判断是否有‘0’，根据行列信号的值判断哪一个按键被按下。模块使用 500kHz 的频率对矩阵键盘进行列扫描，扫描的过程中使用了状态机的思想，即用模块内部定义的 reg 型变量 state 控制键盘扫描的全过程。state=0 时可编程器件向矩阵键盘的四个列均写‘0’（col<=4’b0000），这时任意一列的按键按下都会触发状态机的状态转移，模块开始扫描键盘的每一列以确定按键的位置。state=1 时器件向第 0 列写 0（col<=4’b1110），如果按下的按键位于这一列，则转入判断状态 state=5，否则 state 改变为 2，开始扫描第 1 列，以此类推。通过 state 的转移，模块扫描了键盘的每一列，同时在 state=5 时通过对行列信号是否为 0 的判断确定出按键的位置，输出按下的按键的坐标 row_location 和 col_location。如果没有有效的按键按下，row_location 和 col_location 均赋成 3’b100，这对应在点阵上不是地鼠可能产生的位置，不会对 controller 模块判断地鼠和按键坐标是否对应产生干扰。

（6） 点阵显示模块 led_88

这个模块在初始状态显示绿色边框，游戏状态显示绿色边框和对应(rowran,colran)位置的地鼠，游戏成功状态显示绿色笑脸，游戏失败状态显示红色哭脸。点阵显示图案需要对点阵进行逐行扫描，依次将控制行的信号输出低电平，同时通过调整控制红色发光二极管的列信号和控制绿色发光二极管的列信号使点阵的每一行显示图案的不同组成部分。为此设置一个 3 位宽的计数器 cnt，当 500Hz 时钟（扫描时钟）的上升沿到来时 cnt 加 1，在时钟的驱动下 cnt 从 000 到 111 循环变化，配合

always 块对点阵的各行使能 ROW0~ROW7 依次循环赋 0，再通过对各行控制红色发光二极管的列信号 col_r 和控制绿色发光二极管的列信号 col_g 的不同赋值，使点阵各行显示图案的各行，从而完成整个图案的显示。各行预置好不同的显示图案，通过 controller 传来的 state 控制点阵此时显示的内容。

游戏状态下，当 controller 传来 “打中”信号 hit=1 时，点阵中地鼠消失，直到新的地鼠产生时才会再次显示。为产生这个效果，设置一个变量 hit_tmp，在每只新的地鼠产生时 hit_tmp=0，一旦检测到 hit 的上升沿（代表地鼠被打中）hit_tmp=1，将 hit 的状态保持到 0.5Hz 时钟的有效边沿到来（新地鼠产生）为止。一旦新地鼠产生，hit_tmp 立刻置 0，以此类推。通过在 always 块中对地鼠显示的列 colran 进行译码，译码结果为 col_4。在逐行扫描中首先判断这一行是不是地鼠要显示的那一行(即 cnt==rowran)，如果是，且地鼠没有被打中（!hit_tmp），那么 col_r={2’b00,col_4,2’b00};，否则 col_r=8’h00;完成游戏状态地鼠的显示。

（7） 倒计时模块 timer

一旦复位信号有效或者系统处于初始状态（state==2’b00），time_shi 赋值成 4，time_ge 赋值成 0；进入游戏状态（state==2’b01），timer 开始在 1Hz 时钟上升沿的驱动下进行倒计时，直到计到 00 倒计时停止。在游戏成功状态（state==2’b10）和游戏失败状态(state==2’b11)倒计时停止，时间各位保持当前数字不变。

（8） 蜂鸣器驱动模块 buzzer

为增强游戏效果，我设计了这个模块。当地鼠被打中时蜂鸣器发声，地鼠没被打中时蜂鸣器不响，由 assign beep= en?clk_1a:0;控制。en=1 时蜂鸣器与 880.00Hz 的信号 clk_1a（中音 6）连接，可以发声；en=0 时蜂鸣器与电平‘0’连接，蜂鸣器不发声，从而实现打中地鼠时发声的效果。

（9） 数码管显示译码模块 decoder

对传入的不同 num，decoder 把它译制成不同的 8 位二进制码，使数码管显示 num 对应的数字。

（10） 顶层模块 whack_a_mole

顶层模块例化各个子模块，实现对时间和分数的译码和数码管的段选和位选，使时间和分数用数码管得以显示。

# 3.总体框图

基于题目的设计要求和以上的基本设计思路，系统的总体结构框图和模块划分方框图如图 4 和图 5 所示。

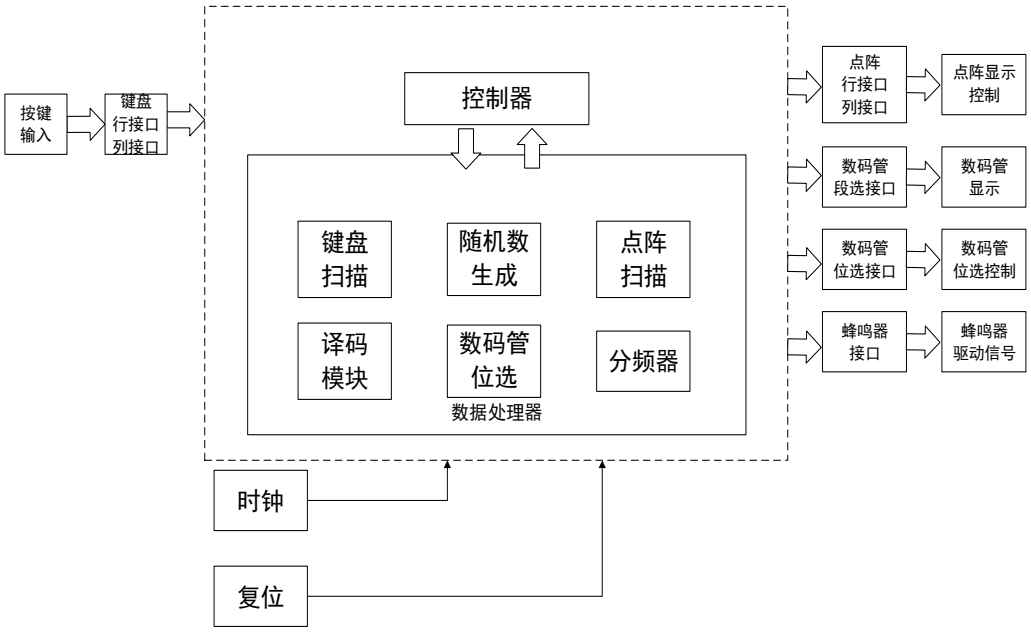


图 4 系统结构框图

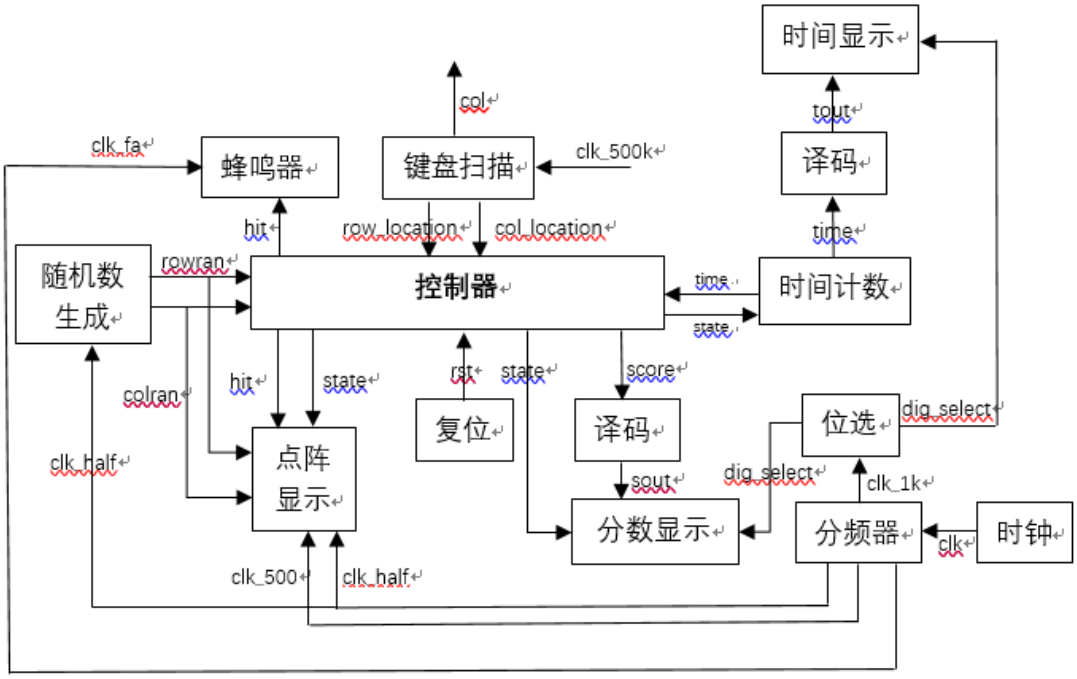


图 5 系统的模块划分方框图

# 三、 仿真波形及波形分析

## 1. controller 模块

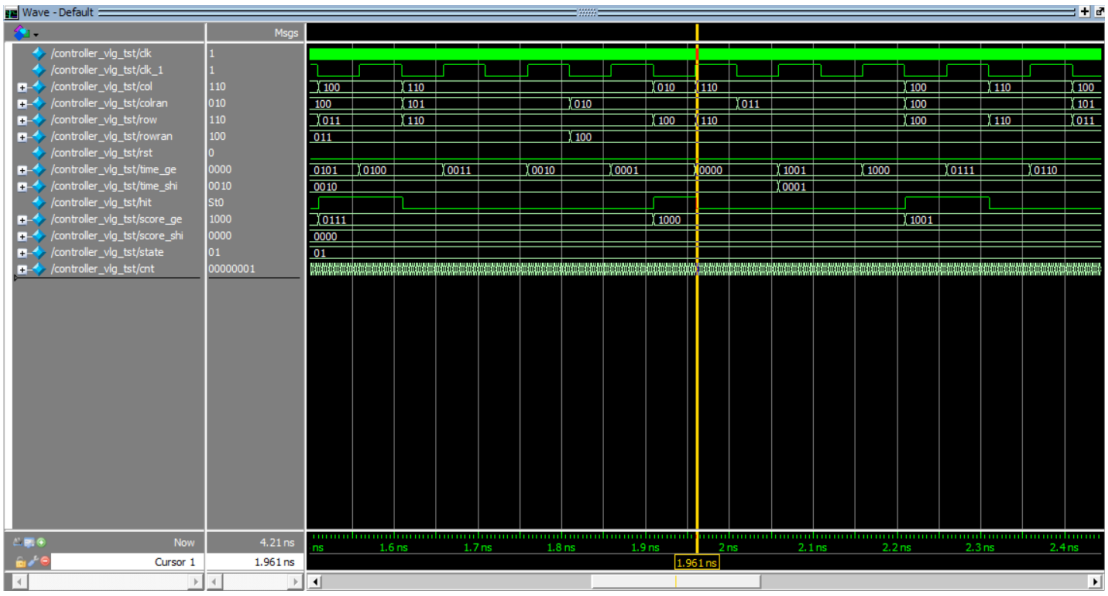


图 6 controller 模块游戏状态仿真波形图

图 6 是 controller 模块游戏状态的仿真波形图。可以看到，输出 state=2’b01，代表当前处于游戏状态。当随机数 rowran，colran 分别和 row，col 相等时（也就是地鼠在点阵上出现的位置和按下的按键对应时），输出 hit=1，同时分数加 1.

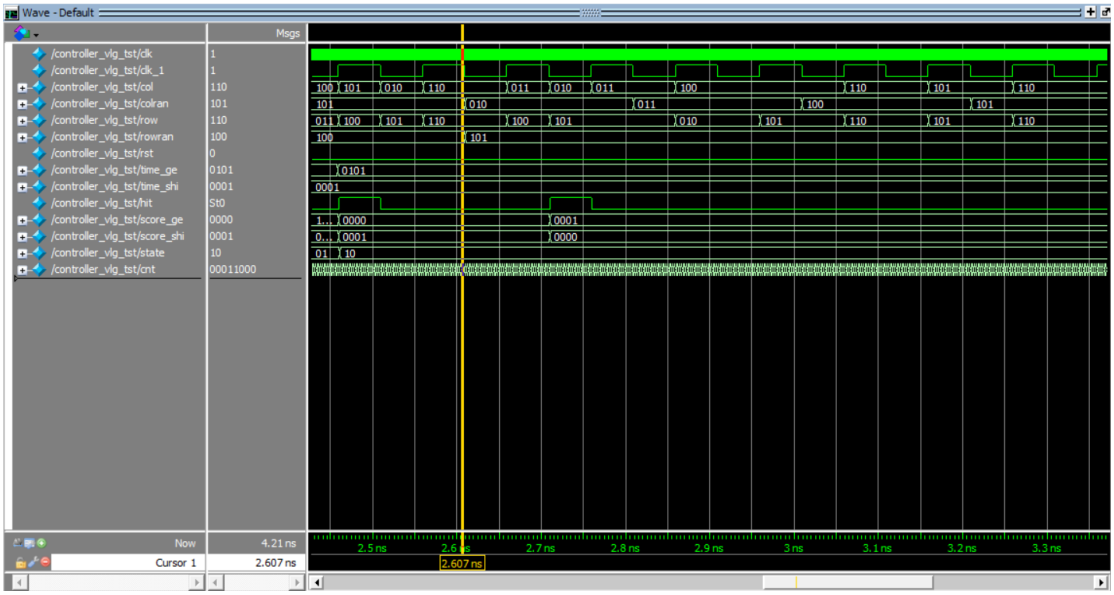


图 7 controller 模块游戏成功状态仿真波形图

图 7 是 controller 模块游戏成功状态的仿真波形图。可以看到，当第十次打中地鼠时，分数输出变为 10 分，同时 state 变为 2’b10，控制各模块进入游戏成功状态。此时倒计时停止，系统的分数显示 10 分。

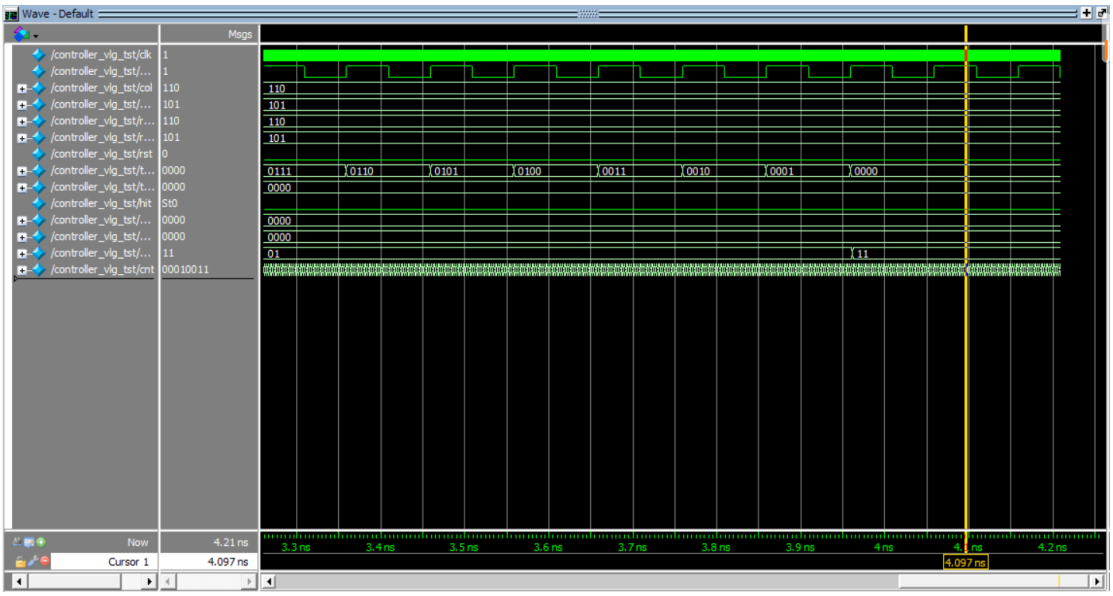


图 8 controller 模块游戏失败状态仿真波形

图 8 是 controller 模块游戏失败状态的仿真波形图。图中时间已到（time_shi==0, time_ge==0），但得分小于 10 分，系统进入游戏失败状态，输出 state=2’b11，倒计时停止。

## 2. fsm 模块


图 9 fsm 模块仿真波形图

图 9 是有限状态机的仿真波形图。当复位键按下时（rst=1），有限状态机（fsm）进入游戏初始状态（state=2’b00）；复位键松开后（rst=0），状态机进入游戏状态（state=2’b01）。在游戏状态中如果 Ts=1，状态机转入游戏成功状态（state=2’b10）；如果 Tt=1，状态机转入游戏失败状态（state=2’b11）；如果 Ts 和 Tt 都为 0，则状态机保持在当前的游戏状态不变（state=2’b01）。

## 3. time_divider 模块

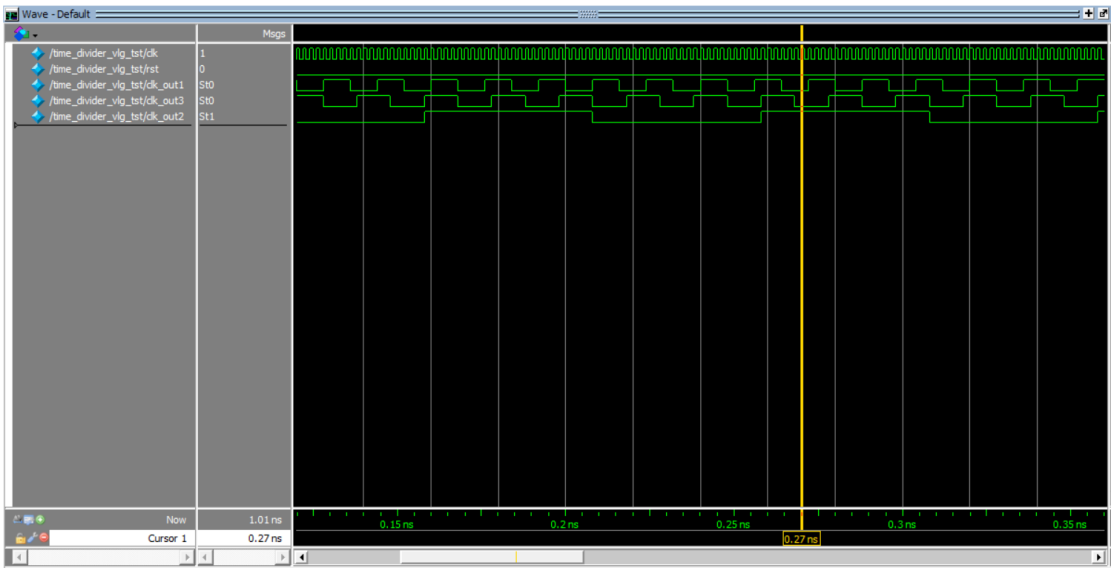


图 10 time_divider 仿真波形图

图 10 是 time_divider 模块的仿真波形图，clk_out1 是对时钟 clk 进行八分频的结果，clk_out3 是对 clk 进行十分频的结果，clk_out2 是对 clk 进行五十分频的结果。

### 4. getrandom 模块

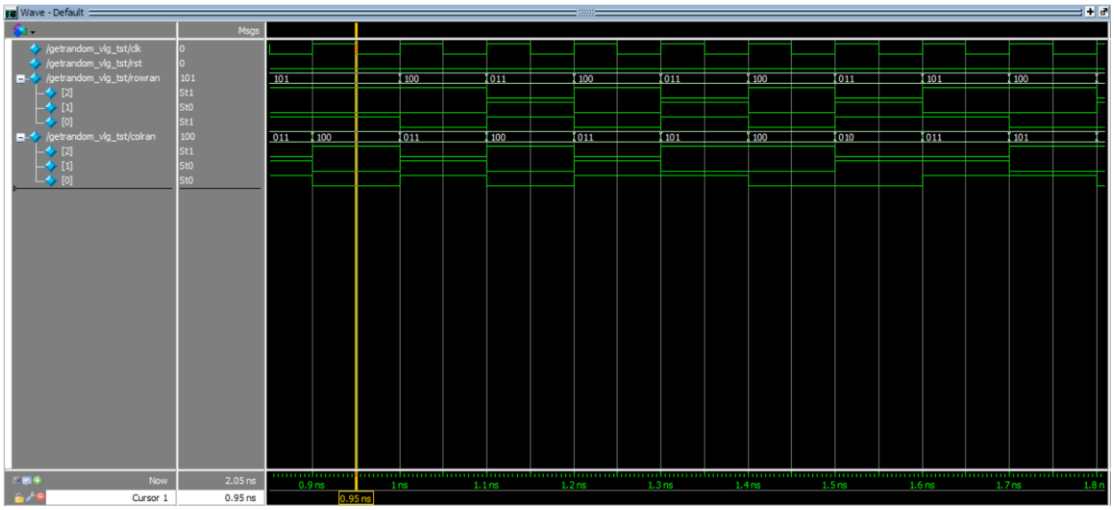


图 11 getrandom 模块仿真波形图

图 11 是 getrandom 模块仿真波形图。在 clk 驱动下，每过 100 个仿真时间单位模块就生成两个新的随机数 rowran 和 colran。

### 5. keyboard_scan 模块

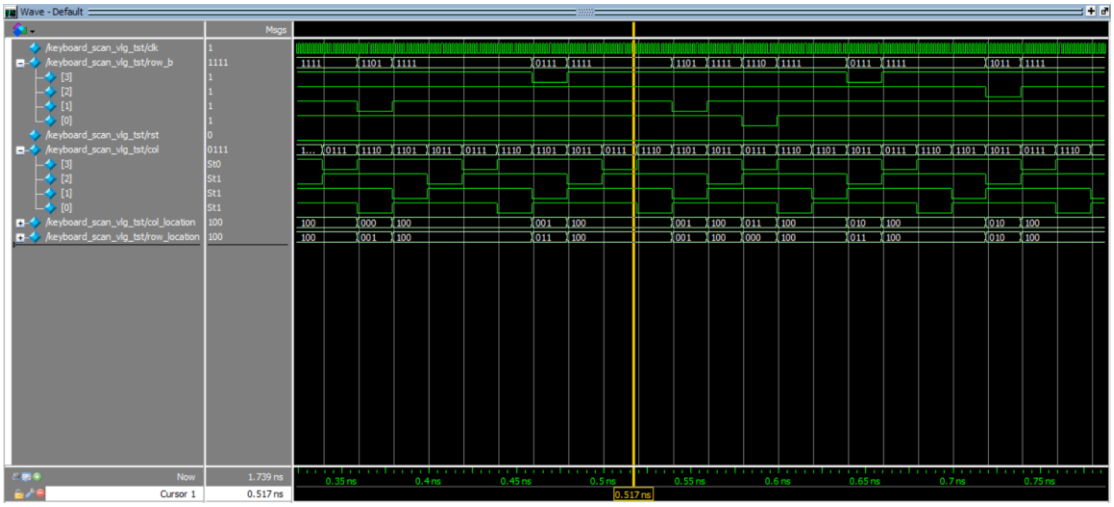


图 12 keyboard_scan 模块仿真波形图

图 12 是 keyboard_scan 模块的仿真波形图。从图中可以看到，系统向矩阵键盘的列信号依次写‘0’，同时读取行是否有 0，根据行列信号的值确定按下的按键的坐标。当按键不被按下时，模块输出的按键坐标为第 4 行，第 4 列，这不是矩阵键盘上的有效按键坐标，不会对判断模块对按键和地鼠位置是否对应的判断产生干扰。图中按下的几个按键的坐标依次为：(1,0)，(3,1)，(1,1)，(0,3)，(3,2)和(2,2)。

## 6. led_88 模块

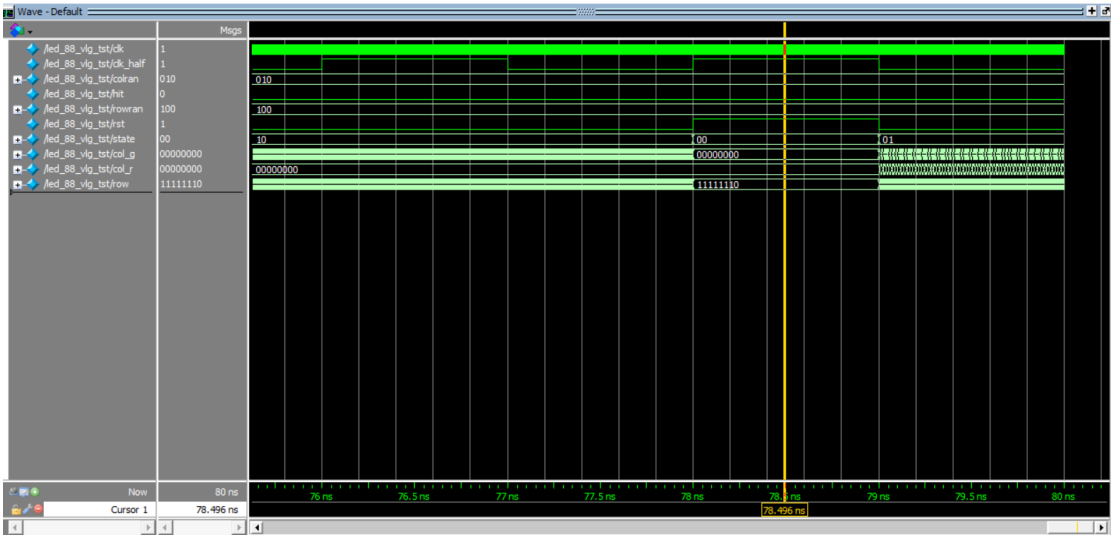


图 13 led_88 模块初始状态仿真波形图

图 13 是初始状态 8×8 点阵的显示情况，state=2’b00。这时控制点阵逐行扫描的计数器 cnt 一直为 0，所以点阵只有第 0 行的内容可以显示。由于第 0 行在初始状态没有设置显示内容，所以控制绿色、红色发光二极管列信号 col_g,col_r 均为全 0.

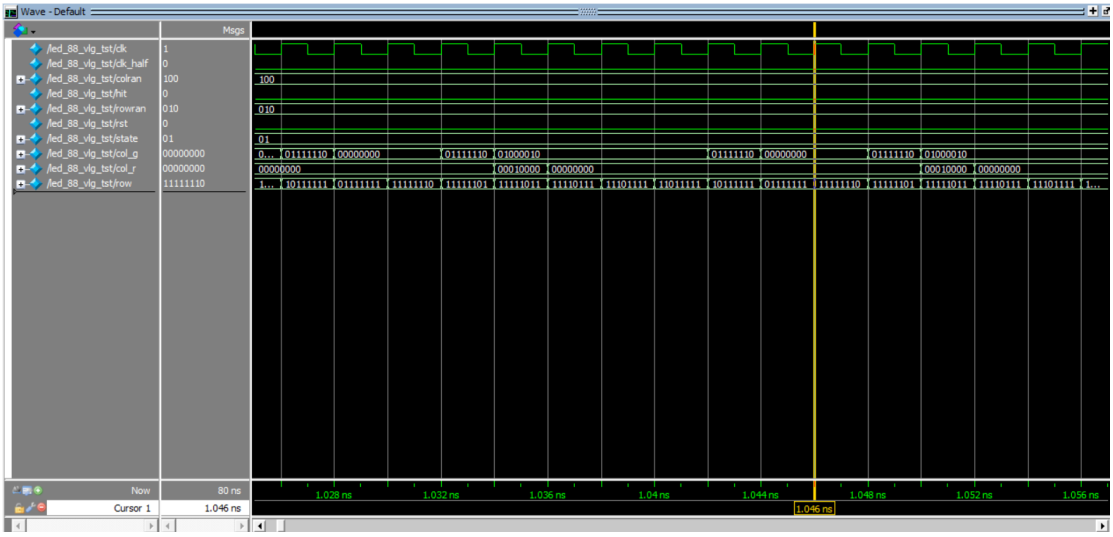


图 14 led_88 游戏状态未击中地鼠时仿真波形图

图 14 是游戏状态下的点阵显示情况，state=2’b01。可以看出，控制点阵各行的信号 row[0]~row[7]依次有效（低电平）。对应不同的行，不同列的显示情况也不同，第 0 和第 7 行 col_g=8’b00000000，绿色 LED 始终不亮；第 1 到 6 行，绿色 LED 显示的图案组成游戏边界。当前 rowran=2,colran=4，对应在第 2 行控制红色 LED 的列信号的值为 col_r=8’b00010000（即控制第 4 列红色发光二极管的列信号有效），配合行信号完成坐标(2,4)的地鼠的显示。对应其他行，col_r 全 0，表示其他行第 4 列的红色 LED 均不亮。

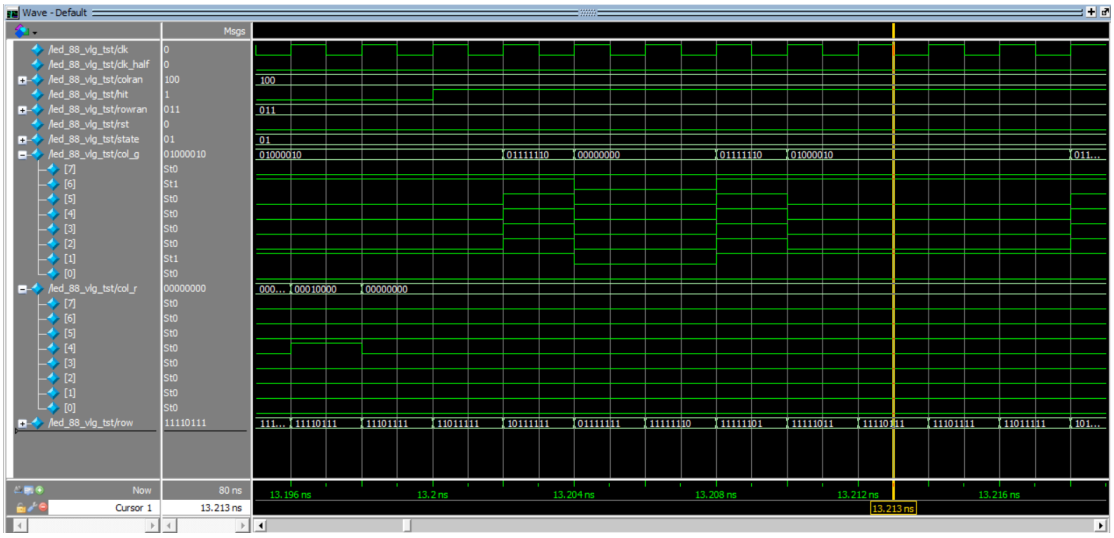


图 15 led_88 游戏状态击中地鼠前后仿真波形图

图 15 是游戏状态下击中地鼠时的仿真波形图。可以看到，当前 rowran=3,colran=4，在 hit=0 时，对应第三行时 col_r=8’b00010000；而当 hit 由 0 变为 1 时，再次扫描到第三行时 col_r=8’b00000000，地鼠不再显示。

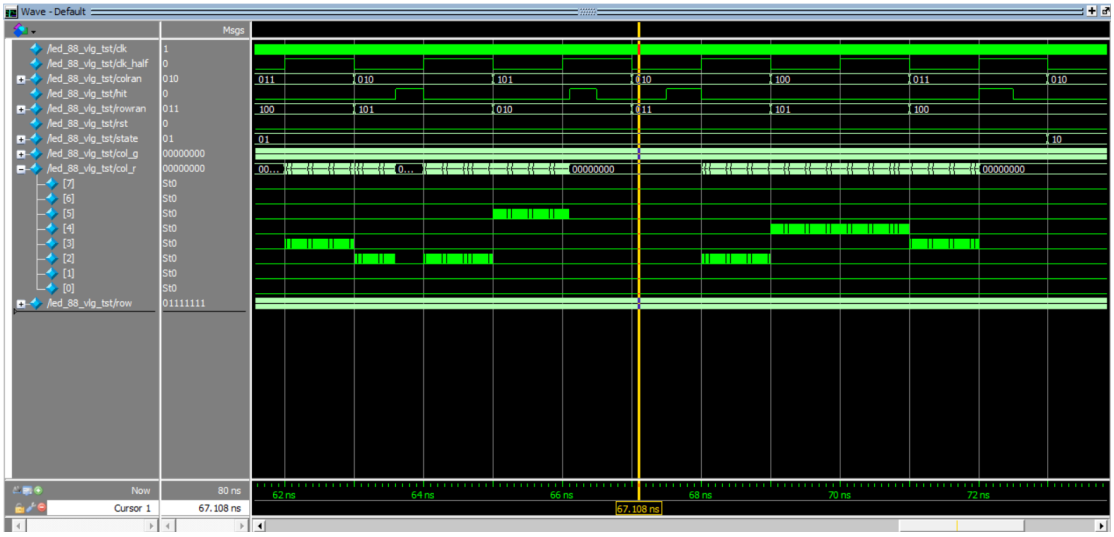


图 16 2s 内两次按键 led_88 仿真波形

图 16 是在 2s 内两次打地鼠时的波形图。可以看到，在第一次打中后（hit=1），col_r 即变为全 0，之后在新地鼠产生之前不论是否按下对应按键（按下按键 hit=1），地鼠都不会再显示。

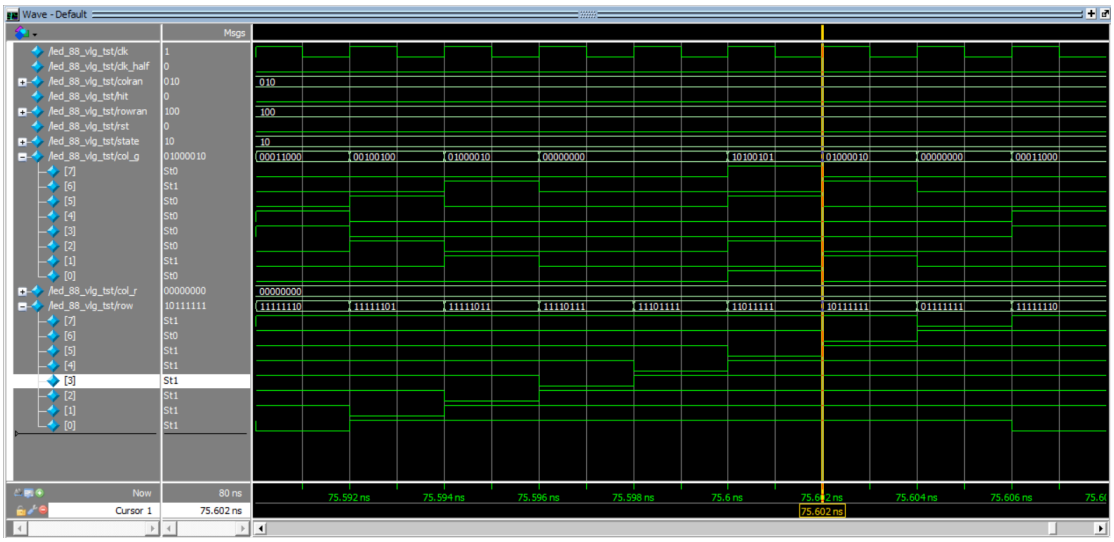


图 17 led_88 游戏成功状态仿真波形图

图 17 是游戏成功状态点阵的显示情况，state=2’b10。可以看到，点阵显示绿色笑脸图案。效果图如下。

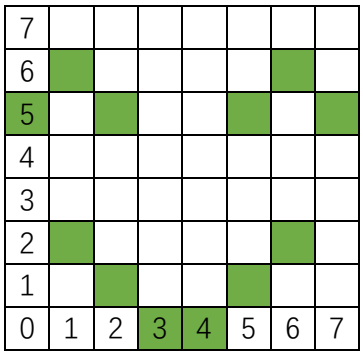


图 18 led_88 游戏成功状态显示效果图

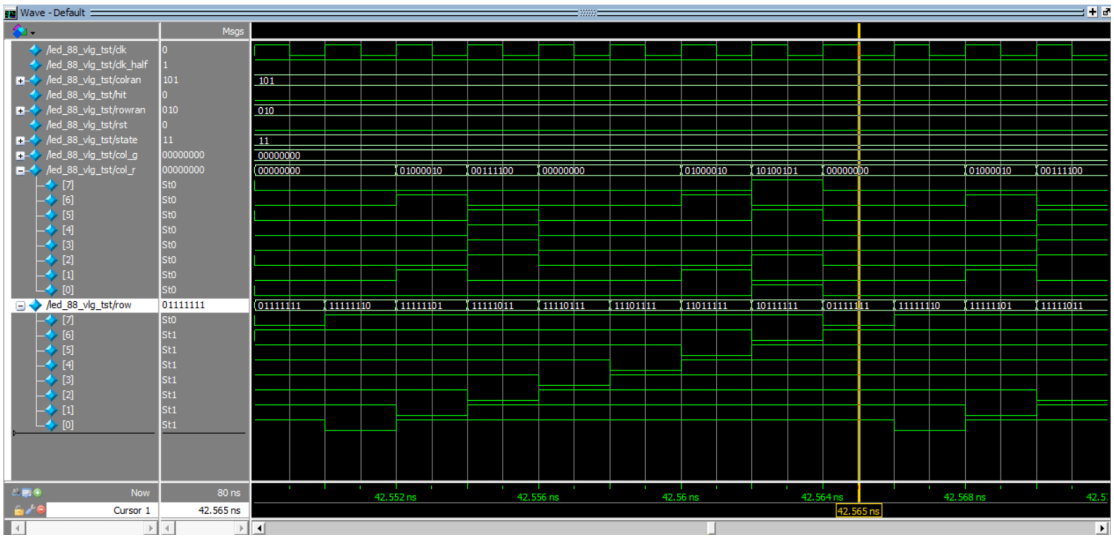


图 19 led_88 游戏失败状态仿真波形图

图 19 为游戏失败状态点阵的仿真波形图,state=2’b11。可以看出此时点阵显示如下的红色哭脸图案。

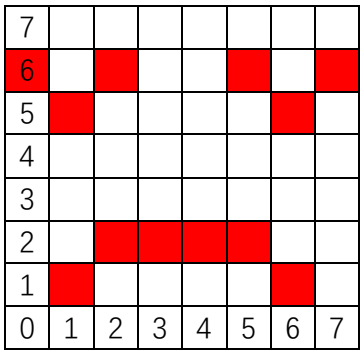


图 20 led_88 游戏失败状态显示效果图

## 7. timer 模块


图 21 timer 游戏成功状态仿真波形图

图 21 是游戏成功时的倒计时仿真波形。可以看到，复位结束后(rst=0)，timer 开始倒计时。当 state=2’b10，系统进入游戏成功状态时，timer 倒计时停止，保持当

前数字不变。

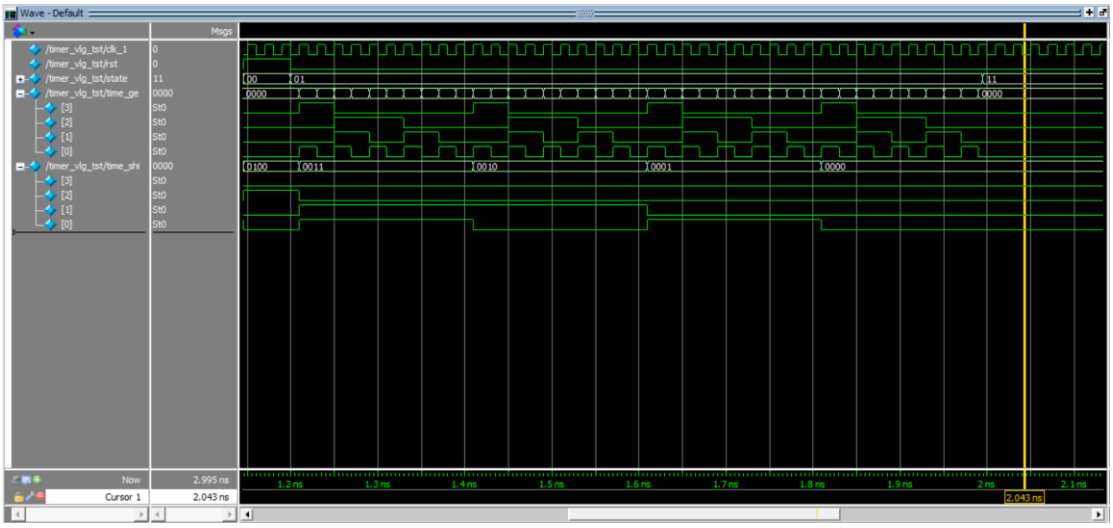


图 22 timer 游戏失败状态仿真波形图

图 22 是游戏失败时的仿真波形。可以看到，当游戏时间超过 39 秒且分数未达到 10 分时，系统进入游戏失败状态。timer 的十位和个位保持 00 不变。

## 8. buzzer 模块

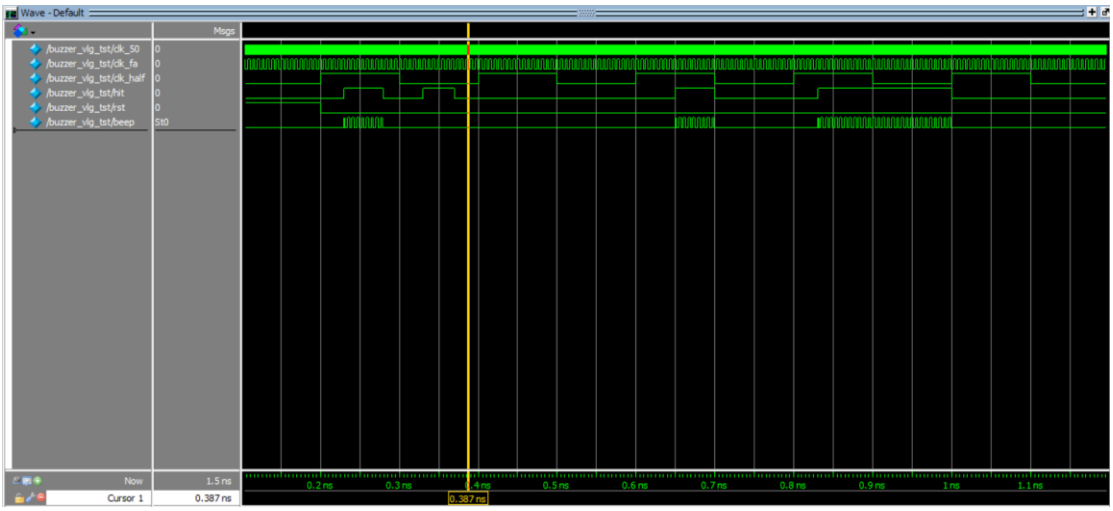


图 23 buzzer 仿真波形图

从图中可以看到，在新的地鼠产生时的第一次击中，也就是 hit 第一次为 1 可以使蜂鸣器发声，发声时间就是按住按键的时间.一旦按键松开后再次按下按键，蜂鸣器不会再次发声。这时点阵上地鼠已经消失，再次按下按键其实没有打中地鼠。两者的现象配合的更好。

## 9. 顶层模块 whack_a_mole 中数码管显示部分的仿真

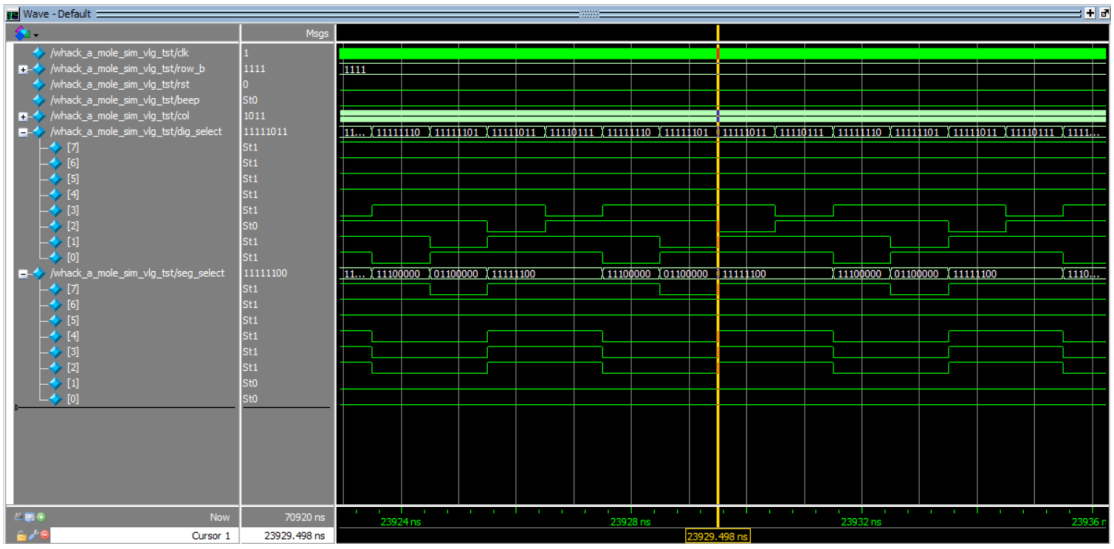


图 24 whack_a_mole 数码管显示部分仿真波形图

图 24 是顶层模块仿真中部分输出信号的波形。为主要体现数码管显示部分的控制情况，图中删去了一些信号的输出。dig_select[7:0]分别控制 8 位 7 段数码管的共阴极端 CAT7~CAT0, seg_select[7:0]分别控制数码管的 AA~AP 控制引脚。从图中可以看出，我使用 8 位 7 段数码管的后 4 位 DISP3~DISP0，所以使 dig_select[7:4]始终为 4’b1111（不能点亮），dig_select[3:0]在 1kHz 时钟上升沿的驱动下在 1110，1101，1011，0111 四种状态中循环，使每次只有 1 位置 0，通过循环扫描完成后四位数码管的显示。对应不同位数码管，各段的显示情况也各不相同。实验中我用数码管 DISP1 和 DISP0 显示倒计时，数码管 DISP3 和 DISP2 显示得分。图中，由于没有打中地鼠，对应分数显示的两位数码管显示“00”（dig_select=8’b11111011 和 8’b11110111 时 seg_select=8’b11111100，是数字“0”）。此时游戏时间剩余 17 秒，对应时间显示的两位数码管显示“17”（dig_select=8’b11111110 时 seg_select=8’b11100000，是数字“7”；dig_select=8’b11111101 时 seg_select=8’b01100000，是数字“1”）。此外 beep=0，蜂鸣器不响。

# 四、 工程代码

```
module whack_a_mole (clk,rst,matrow,matcol_r,matcol_g,
                row_b,col,seg_select,dig_select,beep);        //顶层模块
input clk,rst;                  //系统 50MHz 时钟、复位
input [3:0] row_b;              //读取键盘输入
output [3:0] col;               //键盘列扫描控制
output [7:0] matrow,matcol_r,matcol_g;   //点阵显示输出
output reg [7:0] seg_select;     //数码管段选
output reg [7:0] dig_select;     //数码管位选
output beep;                     //蜂鸣器输出

wire [2:0] colran,rowran;          //随机数模块 getrandom 输出的行随机数和列随机数
wire [2:0] row_location,col_location;   //keyboard_scan 键盘扫描出按键的行列坐标
wire hit;                                //控制器 controller 输出的击打状态
```

```
    wire [3:0] time_shi,time_ge;     //倒计时器 timer 输出的倒计时
    wire [3:0]score_shi,score_ge;   //控制器输出的分数
    wire [1:0] state;                 //有限状态机 fsm 输出的状态
    wire [7:0] seg_time_1,seg_time_0;      //倒计时十位、个位的译码结果
    wire [7:0] seg_score_1,seg_score_0;      //分数十位、个位的译码结果

    wire clk_1k,clk_500,clk_1,clk_half,clk_50,clk_la;
    time_divider #(50000,15) td_1k(clk,rst,clk_1k);  //分频得 1kHz 时钟信号
    time_divider #(2,2) td_500(clk_1k,rst,clk_500); //分频得 500Hz 时钟信号
    time_divider #(10,4) td_50(clk_500,rst,clk_50); //分频得 50Hz 时钟信号
    time_divider #(50,6) td_1(clk_50,rst,clk_1);   //分频得 1Hz 时钟信号
    time_divider #(2,2) td_half(clk_1,rst,clk_half);//分频得 0.5Hz 时钟信号
    time_divider #(56818,15) td_fa(clk,rst,clk_la);//中音 6，880.00Hz
    getrandom gr1(.clk(clk_half),.rst(rst),.ran1(rowran),.ran2(colran));          //生成随机数

    keyboard_scan kb(.clk(clk),.rst(rst),.row_b(row_b),.col(col),
        .row_location(row_location),.col_location(col_location));                  //键盘扫描
    timer
time_counter(.clk_1(clk_1),.state(state),.rst(rst),.time_shi(time_shi),.time_ge(time_ge));  //倒计
时器
    controller
center_control(.rowran(rowran),.colran(colran),.row(row_location),.col(col_location),
        .time_shi(time_shi),.time_ge(time_ge),.clk(clk),.clk_50(clk_50),.clk_half(clk_half),.rst(r
st),.hit(hit),
        .state(state),.score_shi(score_shi),.score_ge(score_ge));
                                    //控制模块和状态机
    led_88
ld(.clk(clk_500),.clk_half(clk_half),.rst(rst),.row(matrow),.col_g(matcol_g),.state(state),
                        .col_r(matcol_r),.rowran(rowran),.colran(colran),.hit(hit));  //点阵显示

    always@(posedge clk_1k or posedge rst)          //数码管位选，以 1kHz 频率扫描
    begin
        if(rst)
        dig_select<=8'b1111_1110;
        else begin
        dig_select[3:0]<={dig_select[2:0],dig_select[3]};
        dig_select[7:4]=4'b1111;
        end
    end
    decoder deco_time_shi(time_shi,seg_time_1);       //对时间十位进行译码
    decoder deco_time_ge(time_ge,seg_time_0);          //对时间个位进行译码
    decoder deco_score_shi(score_shi,seg_score_1);   //对分数十位进行译码
    decoder deco_score_ge(score_ge,seg_score_0);      //对分数个位进行译码
```

```
    always @(dig_select[3:0] or seg_time_0 or seg_time_1 or seg_score_0 or seg_score_1)
//位选:1110 时间个位，1101 时间十位，1011 分数个位，0111 分数十位
    begin
        case(dig_select[3:0])
        4'b1110: seg_select=seg_time_0;
        4'b1101: seg_select=seg_time_1;
        4'b1011: seg_select=seg_score_0;
        4'b0111: seg_select=seg_score_1;
        default: seg_select=8'b0000_0000;
        endcase
    end

    buzzer bz(.clk_la(clk_la),.clk_50(clk_50),.clk_half(clk_half),.hit(hit),.rst(rst),.beep(beep));
//打中地鼠发声

    endmodule

    module buzzer(clk_la,clk_50,clk_half,hit,rst,beep);//蜂鸣器控制模块
        input clk_la,clk_50,clk_half,hit,rst;
        output beep;
        reg en,hit_tmp,flag;
        always@(posedge clk_half or posedge hit)   //储存 hit 状态直到新地鼠产生
        begin
            if(hit) hit_tmp=1;
            else hit_tmp=0;
        end
        always@(posedge clk_50 or posedge rst)
        begin
            if(rst)
            begin
                flag=1;                                    //flag=1 允许改变 en 的值
                en=0;
            end
            else if((hit==1)&&(flag==1)==1)            //打中地鼠之后执行一次
            begin
                en=1;                                      //允许蜂鸣器发声
                flag=0;
            end
            else begin
                if(hit_tmp==0) flag=1;                     //新地鼠产生(hit_tmp=0)后
flag=1，可以在再打中地鼠时响
                else flag=flag;                           //否则 flag 保持 0
                if(hit==0) en=0;                           //按键松开(hit=0)后 en=0，并且由
于 flag 在新地鼠产生之前不变，en 不能置 1
```

```
                else en=en;                                    //按键不松开，蜂鸣器可以继续
发声
            end
        end

        assign beep=en?clk_la:0;
    endmodule

    module
controller(rowran,colran,row,col,time_shi,time_ge,clk,clk_50,clk_half,rst,hit,state,score_shi,score
_ge);    //控制模块
    input [2:0] rowran,colran,row,col;                  //输入 getrandom 模块产生的随机数和
keyboard_scan 模块扫描的行列坐标值
    input clk,clk_50,clk_half,rst;                       //输入时钟和复位
    input [3:0] time_shi,time_ge;                        //输入计时时间值
    output reg hit;                                      //击打状态输出
    output [1:0] state;                                  //输出 fsm 的状态输出
    output reg [3:0] score_shi,score_ge;                //分数输出
    reg Ts,Tt;
    reg flag;
    reg hit_tmp;

    fsm central_fsm(clk,rst,state,Ts,Tt);               //例化有限状态机，完成对整个电路状态
的控制

    always@(posedge clk_50 or posedge rst)
    begin
        if(rst) begin
            hit=0;
        end
        else if(state==2'b01)                           //游戏状态
        begin
            if((rowran==row+2)&&(colran==col+2))        //如果打中，hit 修改为 1
            begin
                hit=1;
            end
            else begin                                  //否则 hit 保持 0
                hit=0;
            end
        end
        else begin                                      //其他情况 hit=0
            hit=0;
        end
    end
```

```
    always @(posedge clk_half or posedge hit) //一旦打中，hit_tmp 赋值 1，并且把这种状态一直记忆到新地鼠产生(0.5Hz 时钟有效边沿到来)
    begin
      if(hit) hit_tmp=1;
      else hit_tmp=0;
    end

    always @(posedge clk_50 or posedge rst)
    begin
        if(rst)                                    //复位，分数置成 00
        begin
          flag=1;
          score_shi=0;
          score_ge=0;
        end
        else if((hit==1)&&(flag==1)==1)          //打中(hit=1)且 flag=1 时才会执行，防止多次打中分数累加的逻辑错误
        begin
            if(score_ge<9)
            begin
              score_ge=score_ge+1;
              score_shi=score_shi;
            end
            else begin
              score_ge=0;
              score_shi=score_shi+1;
            end
            flag=0;                                //分数修改后 flag 即置成 0，防止多次累加
        end
        else begin
          score_ge=score_ge;                       //其他状态分数保持不变
          score_shi=score_shi;
          if(hit_tmp==0) flag=1;                   //直到新地鼠出现后(hit 重新置 0，hit_tmp 也同时置 0)flag 才会恢复 1，可以在打中下一只地鼠时分数加 1
          else flag=flag;                          //新的地鼠不产生，flag 保持不变，使总打一只时分数不能不停增加
        end
    end

    always @(score_shi or rst)
    begin
        if(rst) Ts=0;
        else if(score_shi==1)                              //计到 10 分，Ts=1，其他时候
```

```
Ts=0
        Ts=1;
        else Ts=0;
    end
    always @(time_shi or time_ge or rst)
    begin
        if(rst) Tt=0;
        else if(time_shi==0&&time_ge==0)                    //时间到 Tt=1，否则 Tt=0
        Tt=1;
        else Tt=0;
    end
    endmodule

    module decoder(num,seg_led);//译码器
    input [3:0] num;
    output [7:0] seg_led;
    reg [7:0] seg [9:0];
    initial                             //0~9 对应的数码管译码输出
    begin
        seg[0]=8'b11111100;          //A,B,C,D,E,F,G,DP
        seg[1]=8'b01100000;
        seg[2]=8'b11011010;
        seg[3]=8'b11110010;
        seg[4]=8'b01100110;
        seg[5]=8'b10110110;
        seg[6]=8'b10111110;
        seg[7]=8'b11100000;
        seg[8]=8'b11111110;
        seg[9]=8'b11110110;
    end

    assign seg_led=seg[num];
    endmodule

    module fsm(clk,rst,state,Ts,Tt);          //有限状态机
    input clk,rst,Ts,Tt;
    output [1:0] state;
    reg [1:0] state,next_state;
    parameter state0=2'b00,state1=2'b01,state2=2'b10,state3=2'b11;
    //                          初始状态          游戏状态      游戏成功状态          游戏失
败状态
    always @(posedge clk or posedge rst)        //状态寄存器，时钟有效边沿一到来 state 就
刷新一次，刷新成 next_state 的值
    begin
```

```
        if(rst) state<=state0;                    //如果复位信号有效，state 置成初始状态
        else state<=next_state;
    end

    always @(state or Ts or Tt or rst)
    begin
        case(state)
        state0:begin
        if(!rst) next_state<=state1;          //复位信号失效，转入游戏状态，下一状态是游
戏状态
        else next_state<=state0;               //否则保持初始状态不动
        end
        state1:begin
        if(Ts) next_state<=state2;             //如果分数达到十分，转入游戏成功状态
        else if(Tt) next_state<=state3;       //分数不够的情况下时间到，转入游戏失败状态
        else next_state<=state1;               //否则下一状态仍是游戏状态
        end
        state2:begin
        if(rst) next_state<=state0;           //成功状态下，如果复位转入初始状态
        else next_state<=state2;               //否则保持当前状态
        end
        state3:begin
        if(rst) next_state<=state0;           //游戏失败状态下，如果复位转入初始状态
        else next_state<=state3;               //否则下一状态仍是当前状态
        end
    endcase
    end

    endmodule

    module getrandom(clk,rst,ran1,ran2);//生成随机数，clk 接 0.5Hz

        parameter MIN=2,MAX=5;                //随机数范围 MIN~MAX

    input clk,rst;
    output [2:0] ran1,ran2;
    wire ran_tmp0,ran_tmp1,ran_tmp2,ran_tmp3,ran_tmp4,ran_tmp5;
    wire w,z;
    wire y;

    //移位寄存器组成的 M 序列发生器生成随机数
    D_ff FF0(clk,rst,y,ran_tmp0);
    D_ff FF1(clk,rst,ran_tmp0,ran_tmp1);
    D_ff FF2(clk,rst,ran_tmp1,ran_tmp2);
```

```
    D_ff FF3(clk,rst,ran_tmp2,ran_tmp3);
    D_ff FF4(clk,rst,ran_tmp3,ran_tmp4);
    D_ff FF5(clk,rst,ran_tmp4,ran_tmp5);
    nor nr1(z,ran_tmp0,ran_tmp1,ran_tmp2,ran_tmp3,ran_tmp4,ran_tmp5);
    xor xr1(w,ran_tmp0,ran_tmp5);
    xor xr2(y,w,z);

    assign ran1=2+{ran_tmp4,ran_tmp3}%(MAX-MIN+1);              //生成随机数的范围
2~5
    assign ran2=2+{ran_tmp1,ran_tmp0}%(MAX-MIN+1);

    endmodule

    module D_ff(clk,clr,D,Q);//D 触发器
    input clk,D,clr;              //D 输入，clr 复位
    output reg Q;

    always @(posedge clk)  //主模块下降沿触发，在有效边沿到来前提供稳定随机数
    begin
        if(clr) Q<=1'b0;
        else Q<=D;
    end

    endmodule

    module keyboard_scan(clk,rst,row_b,col,row_location,col_location); //键盘扫描
    input clk,rst;                          //clk 接系统时钟
    input [3:0] row_b;                      //键盘按键按下是 0
    output reg [3:0] col;
    output reg [2:0] row_location,col_location;
    wire [3:0] row;
    reg [2:0] state;
    reg [1:0] cnt;
    wire clk_500k;
    assign row=row_b;
    time_divider #(100,6) td_500k(clk,rst,clk_500k);

    always@(posedge clk_500k or posedge rst)
    begin
        if(rst) begin
            col<=4'b0000;
            state<=0;
            row_location<=3'b100;
            col_location<=3'b100;
```

```
        end
        else begin
        case(state)
        0:
        begin
            col<=4'b0000;                    //四列按键的按下情况都可以感知
            row_location<=3'b100;           //没有检测到有效的按下状态，行列坐标指向一个无效的位置(4,4)
            col_location<=3'b100;
            if(row!=4'b1111)                 //一旦有按键按下，开始逐行检测，通过状态机对整个扫描过程进行控制
            begin
                col<=4'b1110;                  //检测第一行的按键情况
                state<=1;
            end
            else state<=0;
        end
        1:
        begin
            if(row!=4'b1111) state<=5;//如果是这一行的按键被按下，转到状态 5，确定行列坐标
            else begin
                col<=4'b1101;                  //没有按下，开始扫描第二行
                state<=2;
            end
        end
        2:
        begin
            if(row!=4'b1111) state<=5;
            else begin
                col<=4'b1011;
                state<=3;
            end
        end
        3:
        begin
            if(row!=4'b1111) state<=5;
            else begin
                col<=4'b0111;
                state<=4;
            end
        end
        4:
        begin
```

```
                if(row!=4'b1111) state<=5;
                else state<=0;
            end
            5:
            begin
                if(row!=4'b1111)
                begin
                    if(row[0]==0) row_location<=3'b000;        //第 0 行的按键被按下，行坐标
row_location 赋值成 0
                    else if(row[1]==0) row_location<=3'b001;
                    else if(row[2]==0) row_location<=3'b010;
                    else row_location<=3'b011;
                    if(col[0]==0) col_location<=3'b000;
                    else if(col[1]==0) col_location<=3'b001;
                    else if(col[2]==0) col_location<=3'b010;//第 2 列的按键被按下，列坐标
col_location 赋值成 2
                    else col_location<=3'b011;
                    state<=5;
                end
                else state<=0;
            end
            endcase
            end
        end


        endmodule

        module led_88(clk,clk_half,rst,hit,row,col_r,col_g,state,rowran,colran);    //点阵扫描模
块，使用 500Hz 频率扫描
        input clk,clk_half,rst,hit;
        input [2:0] rowran,colran;
        output reg [7:0] row,col_r,col_g;
        input [1:0] state;
        reg [2:0] cnt;
        reg hit_tmp;
        reg [3:0] col_4;

        always @(colran)                    //列随机数转换成对应的地鼠显示情况
        begin
            case(colran)
            3'b010: col_4=4'b0001;          //colran=2，第二列显示地鼠，以此类推
            3'b011: col_4=4'b0010;
            3'b100: col_4=4'b0100;
```

```
            3'b101: col_4=4'b1000;
            default: col_4=4'b0000;
            endcase
        end
        always@(posedge clk_half or posedge hit)   //保持下一组随机数到来时对当前地鼠的击打状态不变
        begin
          if(hit)
          begin
            hit_tmp=1;          //异步时序，保证 controller 传来的 hit 一旦为 1（代表地鼠被打中）,hit_tmp 立刻变为 1，将这种打中状态保持到新地鼠产生为止
          end
          else hit_tmp=0;                          //0.5Hz 时钟上升沿到来，新地鼠产生，默认 hit_tmp=0
        end
        always@(posedge clk)                  //500Hz 频率逐行扫描点阵，cnt=0~7 代表依次使第 0 行到第 7 行可以显示
        begin
            if(rst)      cnt<=0;
            else if(cnt==3'b111) cnt<=0;
            else  cnt<=cnt+1;
        end
        always @(cnt or state or col_4 or rowran or hit_tmp)
        //控制显示不同的图案，初始状态显示游戏边界，游戏状态显示边界和地鼠，游戏成功状态显示绿色笑脸，失败状态显示红色哭脸
        begin
            case(cnt)        //依次选择各行
            3'b000:begin
                row=8'b1111_1110;
                if(state!=2'b10) col_g=8'h00;
                else col_g=8'b0001_1000;
                col_r=8'h00;
            end
            3'b001:begin
                row=8'b1111_1101;
                if(state==2'b00||state==2'b01) col_g=8'b0111_1110;
                else if(state==2'b10) col_g=8'b0010_0100;
                else col_g=8'h00;
                if(state!=2'b11) col_r=8'h00;
                else col_r=8'b0100_0010;
            end
            3'b010:begin
                row=8'b1111_1011;
                if(state!=2'b11) col_g=8'b0100_0010;
```

```
        else col_g=8'h00;
        if(state==2'b01) begin
            if(cnt==rowran&&!hit_tmp) begin
                col_r={2'b00,col_4,2'b00};
            end
            else col_r=8'h00;
        end
        else if(state==2'b11) col_r=8'b0011_1100;
        else col_r=8'h00;
    end
    3'b011:begin
        row=8'b1111_0111;
        if(state==2'b00||state==2'b01) col_g=8'b0100_0010;
        else col_g=8'h00;
        if(state==2'b01) begin
            if(cnt==rowran&&!hit_tmp) begin
                col_r={2'b00,col_4,2'b00};
            end
            else col_r=8'h00;
        end
        else col_r=8'h00;
    end
    3'b100:begin
        row=8'b1110_1111;
        if(state==2'b00||state==2'b01) col_g=8'b0100_0010;
        else col_g=8'h00;
        if(state==2'b01) begin
            if(cnt==rowran&&!hit_tmp) begin
                col_r={2'b00,col_4,2'b00};
            end
            else col_r=8'h00;
        end
        else col_r=8'h00;
    end
    3'b101:begin
        row=8'b1101_1111;
        if(state==2'b00||state==2'b01) col_g=8'b0100_0010;
        else if(state==2'b10) col_g=8'b1010_0101;
        else col_g=8'h00;
        if(state==2'b01) begin
            if(cnt==rowran&&!hit_tmp) begin
                col_r={2'b00,col_4,2'b00};
            end
            else col_r=8'h00;
```

```
            end
            else if(state==2'b11) col_r=8'b0100_0010;
            else col_r=8'h00;
        end
        3'b110:begin
            row=8'b1011_1111;
            if(state==2'b00||state==2'b01) col_g=8'b0111_1110;
            else if(state==2'b10) col_g=8'b0100_0010;
            else col_g=8'h00;
            if(state!=2'b11) col_r=8'h00;
            else col_r=8'b1010_0101;
        end
        3'b111:begin
            row=8'b0111_1111;
            col_g=8'h00;
            col_r=8'h00;
        end
endcase
end

endmodule

module time_divider #(          //分频器
parameter N=2,
        WIDTH=8)
(
        input clk,
        input rst,
        output reg clk_out);        //N 是分频比，WIDTH 计数器位宽
reg [WIDTH-1:0] cnt;
always @(posedge clk or posedge rst)
begin
        if(rst) begin
            cnt<=0;
            clk_out<=0;
        end
        else if(cnt==(N>>1)-1)  //偶分频，计数器计到 N/2-1 取反
        begin
            clk_out<=~clk_out;
            cnt<=0;
        end
        else cnt<=cnt+1;
end
endmodule
```

```
module timer(clk_1,state,rst,time_shi,time_ge);//倒计时器
input clk_1,rst;
input [1:0] state;
output reg [3:0] time_shi,time_ge;

always@(posedge clk_1 or posedge rst)
begin
    if(rst)                                    //复位，重新从 40s 开始倒计时
    begin
        time_shi<=4'b0100;
        time_ge<=4'b0000;
    end
    else if(state==2'b00)                      //初始状态，时间保持 40
    begin
        time_shi<=4'b0100;
        time_ge<=4'b0000;
    end
    else if(state==2'b01)                      //开始倒计时
    begin
        if(time_ge==0&&time_shi!=0)
        begin
            time_ge<=4'b1001;
            time_shi<=time_shi-1;
        end
        else if(time_ge==0&&time_shi==0)   //时间到，数字保持 00 不变
        begin
            time_ge<=time_ge;
            time_shi<=time_shi;
        end
        else time_ge<=time_ge-1;
    end
    else begin                                 //游戏成功、失败状态，倒计
时保持当前数字不变
        time_ge<=time_ge;
        time_shi<=time_shi;
    end
end

endmodule
```

# 五、 功能说明及资源利用情况

本设计可以完成题目的所有基础要求，另外还增加了打中地鼠时发声的功能，增

加游戏的趣味性。

本工程共利用 EPM1270T144C5 芯片的 273 个逻辑单元（logic elements），占芯片逻辑单元总数的 21%。共使用了 51 个管脚（pins），占芯片管脚总数的 44%。

# 六、 故障及问题分析

1. 第三周时可以实现复位后游戏界面正常显示，但是一旦按下和地鼠对应的按键，分数不停跳变且点阵直接显示绿色笑脸或红色哭脸的情况。原因在于 controller 模块中对键盘按键和点阵对应位置检测的 always 块的逻辑判断条件有漏洞。时钟上升沿一到，两位置一旦对应，逻辑判断即为真，分数也就不停累加。解决方法是增加一个 reg 型变量 flag，将加分的逻辑判断条件改为(hit==1)&&(flag==1)==1，一旦加分后 flag 置 0，且设置只有 hit 恢复 0 时 flag 才能被置 1，否则 flag 保持 0.这样按键不松开的情况下分数只会加 1 分，解决了这一问题。
2. 第四周时出现按下按键，打中地鼠，点阵中地鼠消失、分数加 1，但是再按按键分数还会加 1，可是此时地鼠已经消失，游戏现象出现错误。这是因为在 always 块的逻辑判断中是一旦按下正确按键且 flag==1，分数就会加 1。由于这时随机数没有变化，所以按下刚才的按键 always 块的逻辑判断中认为还是按下了正确的按键。而且由于按键松开后 flag 已经置 0，所以分数就可以加 1.解决方法是增加一个以 hit 上升沿和 clk_half(0.5Hz 时钟)上升沿作为敏感列表触发的 always 块。每次新地鼠产生，hit_tmp 都会置 0；一旦打中，hit_tmp 就会置 1，直到新地鼠产生时 hit_tmp 的值才会发生变化。这样就等效于“保持”了地鼠被打中的状态，即使再次按键，对 hit_tmp 的值也不会产生影响。再将 flag 置 1 的条件改为 hit_tmp==0，所以新的地鼠产生前 flag 会保持 0，这样就避免了错误。点阵显示模块和蜂鸣器模块中都采用了这种“状态保持”的思想。
   蜂鸣器的处理更加复杂些，为了只在第一次按下按键时发声，除了由 hit_tmp 控制 flag 的变化外，还由 flag 和 hit 同时控制 en（蜂鸣器发声使能）的变化，即一次打中蜂鸣器只可能响一次，响的时间由 hit=1 的保持时间决定，能不能响由 flag 和 hit 共同决定。hit=1 时 en=1，蜂鸣器可以持续发声；一旦 hit=0，en 就被置 0，蜂鸣器不再发声，而且在新地鼠产生之前 en 没有再变为 1 的机会， 蜂鸣器不会再次发声。这样实现了前述功能。
3. 对于提高要求的两只地鼠，重点在于两只地鼠在同一行，特别是同一个位置的显示。由于 getrandom 生成的随机数不可预知，所以有可能出现两只地鼠在同一个位置的情况。我最后没有让这种情况下按按键两只地鼠同时消失（红色 LED 熄灭）且分数同时加 2 分，所以没有完成提高要求。以后如果有时间，我有兴趣继续研究一下这种情况的处理。

# 七、 总结和结论

通过这次数字系统设计实验，我对自顶向下设计数字系统的过程有了深刻的体会

和认知，对数电理论课学到的基本逻辑单元电路也有了更加深刻的理解。

实验时间是有限的，为了完成实验我需要利用课下的时间提前准备相应的代码模块，这锻炼了我的项目整体规划能力。本实验用到了板子的许多外设，点阵、数码管、矩阵键盘等等，为了能正确地使用它们我需要查找资料，这也锻炼了我的自主学习能力。同时遇到问题小组成员共同讨论、互相请教，这也培养了我们的团队协作的能力。实验中我自己编写 Verilog 代码，并且查阅资料时需要搞懂别人的代码各部分的意思和逻辑，思考他们为什么这么写。在这个过程中我使用硬件描述语言描述和设计数字系统的能力有了很大的提高。

虽然做实验很辛苦，但是通过实验我的各方面能力都得到了提升，以及最后打地鼠游戏的运行成功，这些令我十分兴奋，做实验的疲惫也一扫而光，成就感油然而生。数电实验是一次很宝贵的经历，在过程中我得到了很多，期待还有这样的机会让我能够锻炼自己。