FPGA打地鼠项目文档

简介

项目目的和背景

`FPGA`打地鼠项目是一款基于`FPGA`技术的游戏，能够提高玩家的反应能力和手眼协调能力。该项目是由`Xilinx xc7a35tfgg484-2`开发板和一块扩展的硬件键盘组成，玩家通过按下按钮打击屏幕上出现的地鼠。

本项目旨在设计一个`FPGA`打地鼠游戏，设计了`16`个地鼠以及对应的`16`个键盘输入。通过该游戏的设计和实现，深入了解`FPGA`硬件开发的流程和方法。

开发者是第一次使用`FPGA`硬件进行实际的实现，在这之前只使用过`Verilog`制作一个信号灯的变化逻辑并进行`simulation`验证，所以在开发的过程中有诸多不算规范的地方，而且受限于硬件条件，很多希望实现的功能都没有实现。但是所幸项目开发者有着一些其他项目开发的经验，所以在项目中留下了很多便于扩展的结构，如果有其他可用的硬件设备可以尝试去连线扩展并实现。比如为拓展可以播放`BGM`的外设与击中地鼠的音效反馈留下了一些方便修改的代码结构。

系统架构概述

本项目采用基于`ALINX AX7035`开发板的`FPGA`打地鼠游戏系统。项目采用了`Top-down`的设计流程方法。

1. 首先需要确定游戏的玩法，游戏的场景游戏的难度等等。这个步骤是整个设计的基础，确保需求明确并且明确每个功能所绑定的可交互硬件资源。
2. 分析游戏功能并进行分层：设计状态机的跳转的行为，对不同的功能进行分频的设计。
3. 设计顶层模块，搭好项目整体的接口和架构，针对需求与功能的划分对接口进行细微的调整。
4. 设计各个子模块，包括定时器、游戏逻辑、显示模块、与地鼠交互的接口模块等。每个模块应该尽可能独立且功能设置合理。
5. 进行模块级仿真，进行模块级仿真，以确保每个子模块的功能正确。
6. 由于硬件的限制，只进行简单的系统级仿真，保证系统能正常跳转，不要出现显示的未知状态以及错误的状态跳转即可。

7. 进行综合和实现，将设计综合成 FPGA 能够实现的逻辑电路，然后进行实现，在`Xilinx`的软件上尽可能解决存在的`Warning`，合并冗余的逻辑。
8. 进行时序约束，在实现后，需要进行时序约束，以保证设计的时序正确。
9. 进行时序分析，进行时序分析，以确保实现后的时序符合设计要求。
10. 进行性能优化，对设计进行性能优化，以提高游戏的运行速度和响应速度

通过上面的流程基本完成了游戏的控制模块的设计，能够通过硬件进行进行综合验证。通过输入设备获取玩家的操作指令，控制游戏的进行；游戏显示模块将游戏的状态和图形显示到数码管上或者`LED`灯上，给予玩家操作带来的反馈。整个系统由`FPGA`控制，通过输入输出设备与玩家进行交互。

硬件设计

开发板概述

开发板型号

本项目所使用的开发板型号为`ALINX AX7035`，该开发板采用`Xilinx Artix-7 XC7A35T FPGA`芯片，具有丰富的外设接口和高性能的处理能力。

开发板特性

- `FPGA`芯片：`Xilinx Artix-7 XC7A35T`
- `DDR3`内存：`256MB`
- `Flash`：`128MB`
- 以太网接口：`10/100/1000M`自适应
- `USB`接口：`2`个
- `GPIO`接口：`8`个
- `VGA`接口：`1`个
- `HDMI`接口：`1`个

一些参数设置如下所示：

本项目使用到的硬件资源如下所示：

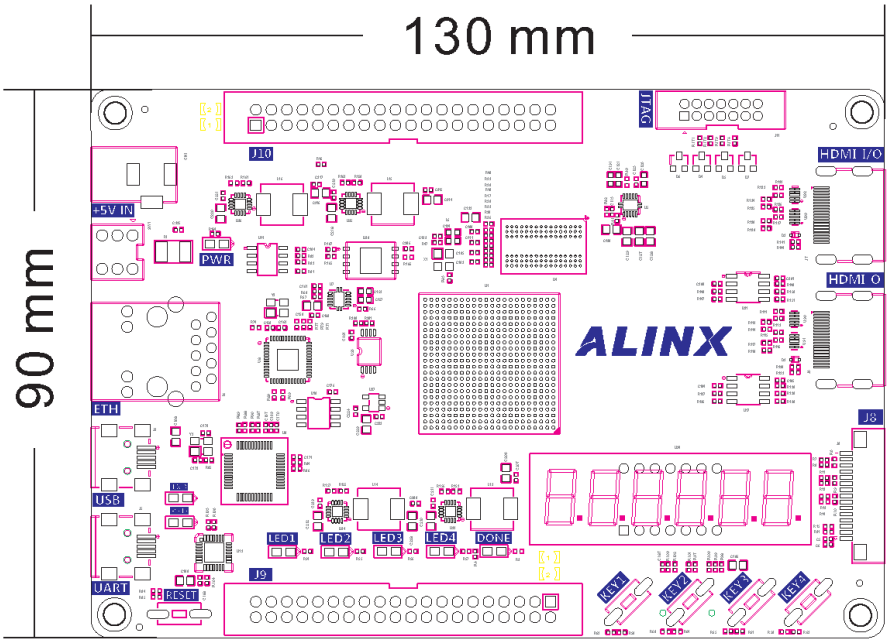
变量名称	硬件**IO**接口	标准电压
clk	PACKAGE_PIN Y18	LVCMOS33

变量名称	硬件**IO**接口	标准电压
rst_n	PACKAGE_PIN F20	LVCMOS33
seg[0]	PACKAGE_PIN J5	LVCMOS33
seg[1]	PACKAGE_PIN M3	LVCMOS33
seg[2]	PACKAGE_PIN J6	LVCMOS33
seg[3]	PACKAGE_PIN H5	LVCMOS33
seg[4]	PACKAGE_PIN G4	LVCMOS33
seg[5]	PACKAGE_PIN K6	LVCMOS33
seg[6]	PACKAGE_PIN K3	LVCMOS33
seg[7]	PACKAGE_PIN H4	LVCMOS33
segIndex[0]	PACKAGE_PIN M2	LVCMOS33
segIndex[1]	PACKAGE_PIN N4	LVCMOS33
segIndex[2]	PACKAGE_PIN L5	LVCMOS33
segIndex[3]	PACKAGE_PIN L4	LVCMOS33
segIndex[4]	PACKAGE_PIN M16	LVCMOS33
segIndex[5]	PACKAGE_PIN M17	LVCMOS33
led_react[0]	PACKAGE_PIN F19	LVCMOS33
led_react[1]	PACKAGE_PIN E21	LVCMOS33
led_react[2]	PACKAGE_PIN D20	LVCMOS33
led_react[3]	PACKAGE_PIN C20	LVCMOS33
mouse[0]	PACKAGE_PIN B15	LVCMOS33
mouse[1]	PACKAGE_PIN B16	LVCMOS33
mouse[2]	PACKAGE_PIN B17	LVCMOS33
mouse[3]	PACKAGE_PIN B18	LVCMOS33
mouse[4]	PACKAGE_PIN A18	LVCMOS33
mouse[5]	PACKAGE_PIN A19	LVCMOS33
mouse[6]	PACKAGE_PIN C18	LVCMOS33
mouse[7]	PACKAGE_PIN C19	LVCMOS33
mouse[8]	PACKAGE_PIN C13	LVCMOS33
mouse[9]	PACKAGE_PIN B13	LVCMOS33
mouse[10]	PACKAGE_PIN A13	LVCMOS33
mouse[11]	PACKAGE_PIN A14	LVCMOS33

变量名称	硬件**IO**接口	标准电压
mouse[12]	PACKAGE_PIN C14	LVCMOS33
mouse[13]	PACKAGE_PIN C15	LVCMOS33
mouse[14]	PACKAGE_PIN A15	LVCMOS33
mouse[15]	PACKAGE_PIN A16	LVCMOS33
key_in[0]	PACKAGE_PIN E13	LVCMOS33
key_in[1]	PACKAGE_PIN E14	LVCMOS33
key_in[2]	PACKAGE_PIN D14	LVCMOS33
key_in[3]	PACKAGE_PIN D15	LVCMOS33
key_out[0]	PACKAGE_PIN E16	LVCMOS33
key_out[1]	PACKAGE_PIN D16	LVCMOS33
key_out[2]	PACKAGE_PIN F13	LVCMOS33
key_out[3]	PACKAGE_PIN F14	LVCMOS33
controlkey[0]	PACKAGE_PIN M13	LVCMOS33
controlkey[1]	PACKAGE_PIN K14	LVCMOS33
controlkey[2]	PACKAGE_PIN K13	LVCMOS33
controlkey[3]	PACKAGE_PIN L13	LVCMOS33

开发板外观

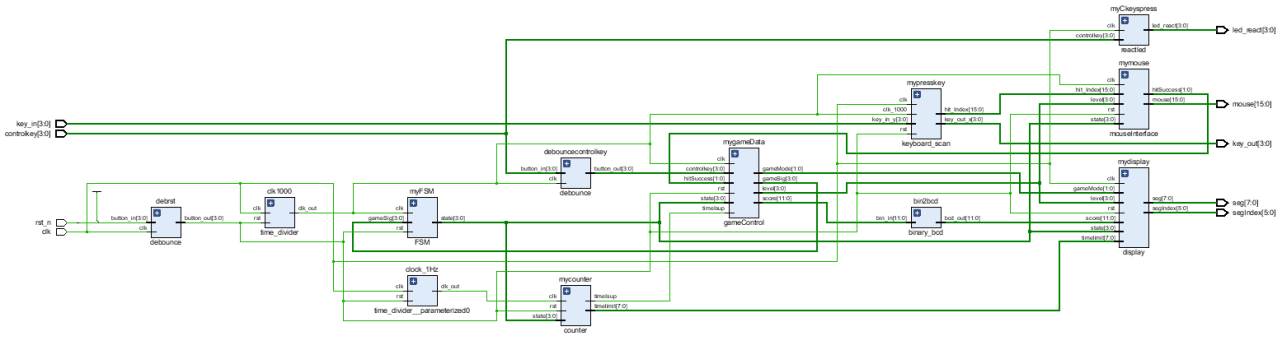
以下是`ALINX AX7035`开发板的尺寸图：



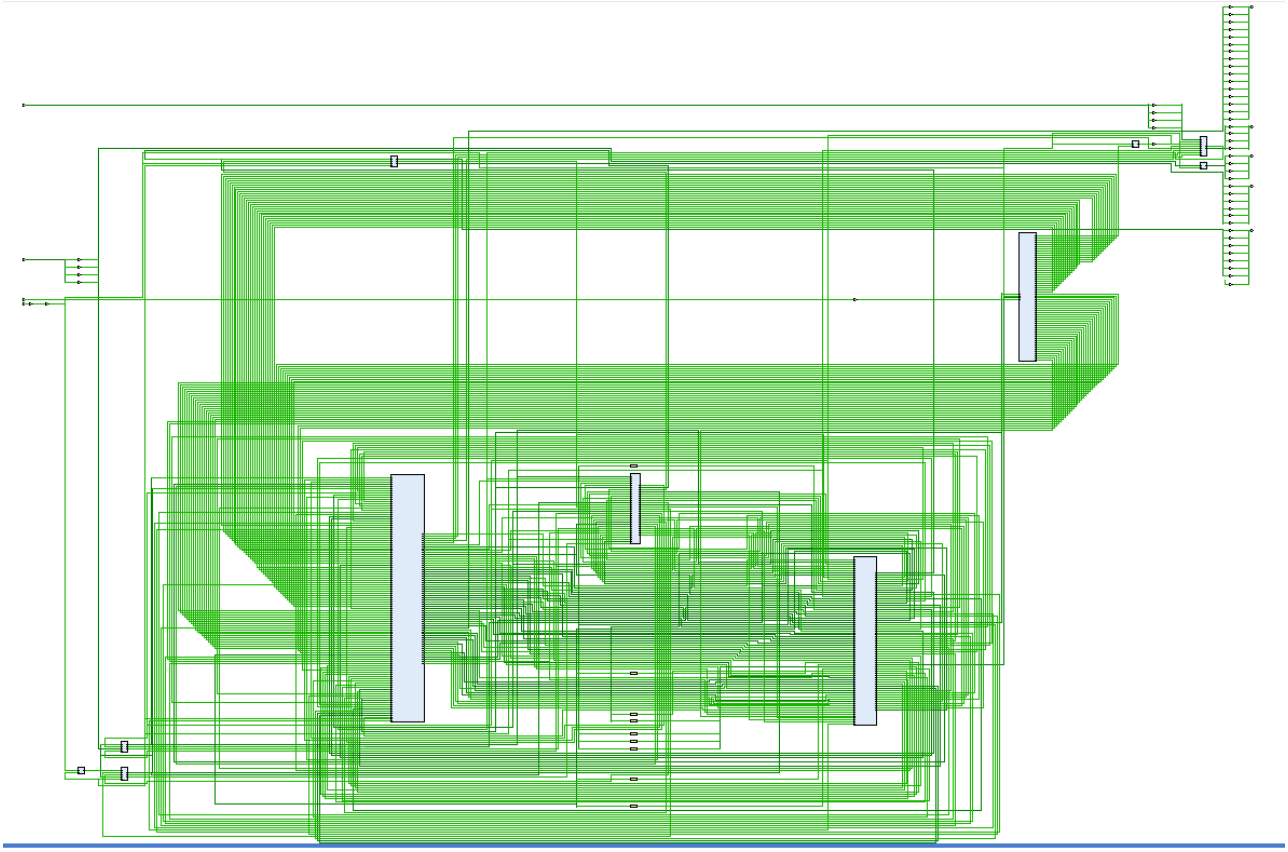
电路设计

电路原理图

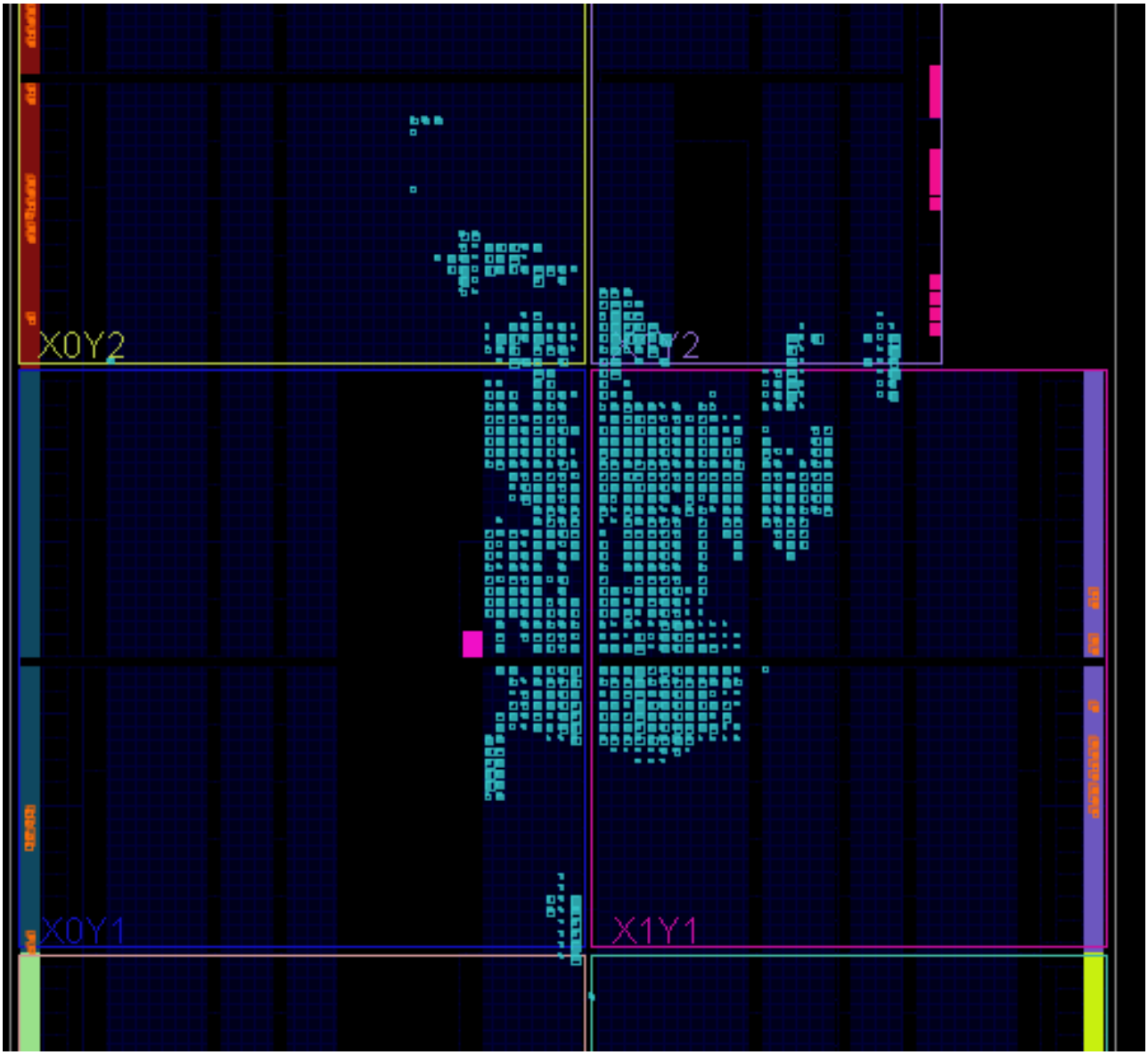
RTL结果图：



综合结果



物理映射结果



电路设计说明

本项目的电路设计主要包括输入设备的接口电路和输出设备的接口电路。其中输入设备的接口电路包括按键输入电路和时钟电路；输出设备的接口电路包括VGA显示电路和音频输出电路。

顶层模块

```
module Top_module (
    input [3:0] controlkey;  // 控制按键
    input [3:0] key_in;  // 键盘输入
    input rst_n, clk;  // 复位，时钟
    output [3:0] led_react;  // 控制按键反馈
    output [15:0] mouse;  // 地鼠
    output [7:0] seg;  // 数码管
    output [5:0] segIndex;  // 数码管选通
    output [3:0] key_out;  // 键盘选通
);
```

接口定义

输入接口定义

`controlkey`：游戏设置按键

`key_in`：打地鼠按键，与`key_out`共同确定地鼠坐标

`rst_n`：复位按键

`clk`：时钟

输出接口定义

`led_react`：按键反馈灯

`mouse`：地鼠，16个灯

`seg`：7段数码管加一个小数点

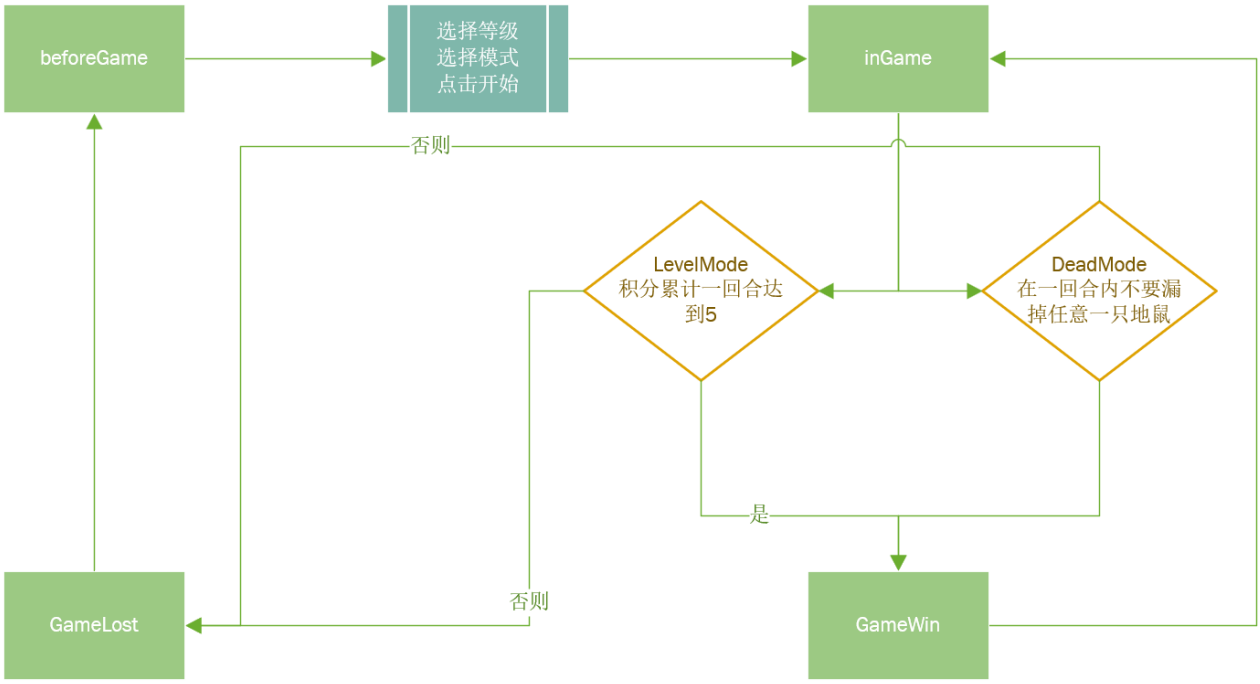
`segIndex`：6个7段数码管的选通信号

`key_out`：打地鼠按键，与`key_in`共同确定地鼠坐标

软件设计

系统流程

本项目的系统流程如下：



项目结构组成

项目文件列表

以下是本项目的文件结构示意：

```
├── reference # 项目参考文件目录
├── src
|   ├──`top_module.v` # 顶层模块
|   ├──`gameControl.v` # 游戏控制逻辑模块
|   ├──`FSM.v` # 游戏状态机模块
|   ├──`display.v` # 游戏显示模块
|   ├──`keyboard_scan.v` # 扩展键盘控制模块
|   ├──`reactled.v` # 按键反馈模块
|   ├──`counter.v` # 计时器模块
|   ├──`mouseIntserface` # 地鼠的控制逻辑接口
|   ├──`debounce.v` # 按键消抖模块
|   ├──`time_divider.v` # 时钟分频模块
```

```
|   ├──`bin2bcd.v` # 二进制码转换
|   ├──`random.v` # 随机数生成
|   ├──`smg_interface.v` # 数码管接口
|   ├──`smg_scan_modeule.v` # 数码管扫描
|   ├──`smg_encode_module.v` # 数码管编码
|   ├──`smg_control_module.v` # 数码管控制
├── README.md # 项目文档
```

参考资料说明

项目的参考资料中比较重要的是开发板的用户手册以及开发板的示例代码。本项目对这些已有的代码和工具进行了优化和改进，结合项目本身的分频的特点应用到了项目当中。

技术细节

状态机设计与跳转

状态机的设计思路

本项目的状态机设计主要包括游戏状态的定义和状态转移条件的确定。游戏状态包括准备状态、游戏状态和结束状态，状态转移条件包括按键输入、计时到达等条件。项目开发者根据自己的想法，将状态机简单地分为了四个状态。

状态机编码：

```
parameter
beforeGame = 4'b0001,
inGame = 4'b0010,
GameLost = 4'b0100,
GameWin = 4'b1000;
```

1. `beforeGame`：进入游戏前的状态，这个状态可以调整游戏模式，设置游戏开始的等级。
2. `inGame`：游戏中的状态，在这个状态中，会根据当前等级和游戏模式按照预定的逻辑运行。
3. `GameWin`：游戏胜利的状态，在这个状态中，玩家无法设置等级，游戏等级会在进入下一轮游戏时自动加一，分数累计。

4. `GameLost`：游戏失败状态，这个状态中玩家会看到得分闪烁，得分清零，只能回到`beforeGame`状态。

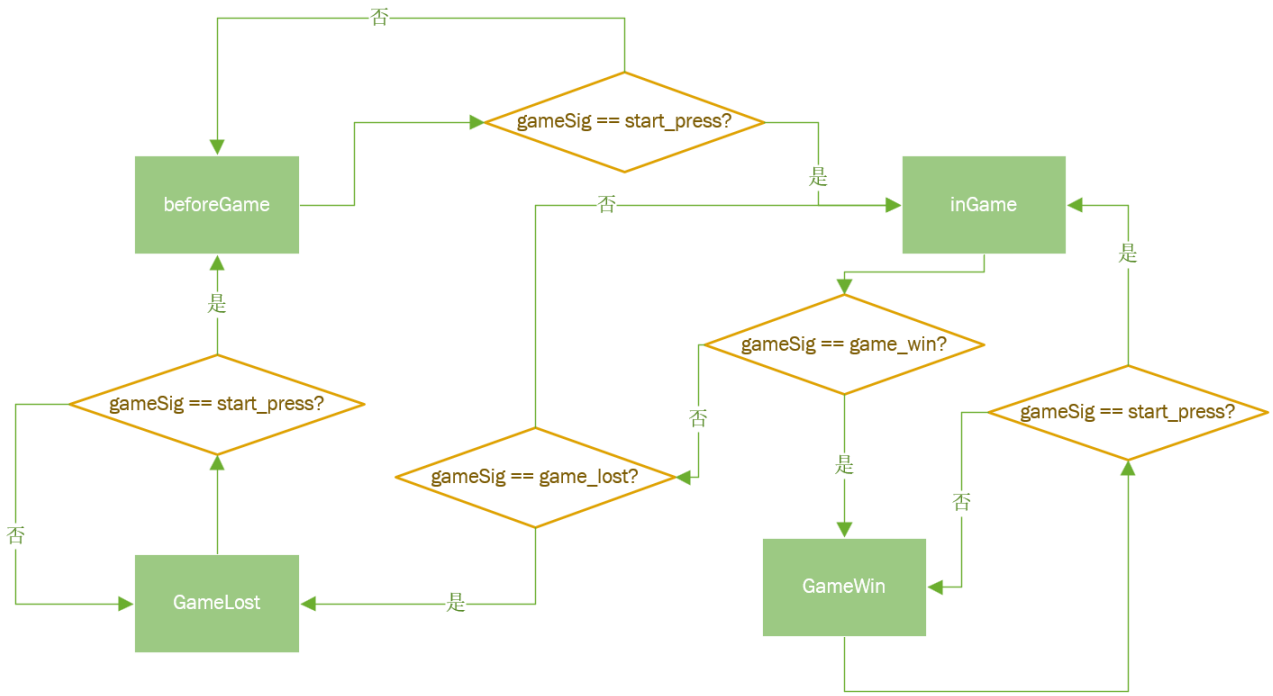
状态转移图

状态跳转信号`gameSig`编码：

```
parameter
keepCurrent = 4'b0001,
game_win = 4'b0010,
start_press = 4'b0100,
game_lost = 4'b1000;
```

状态机得到状态跳转信号之后会进行相应的状态变化或者状态保持。

以下是本项目的状态转移图：



游戏控制逻辑

模块接口

```
// 系统时钟1/1000频率
module gameControl (
    clk,
    rst,
    state,
    hitSuccess,
    timeIsup,
    controlkey,
    level,
    gameMode,
    gameSig,
    score
);
```

模块实现了输出状态跳转信号，存储得分情况，存储游戏等级、游戏模式。

计分逻辑实现

本项目的计分逻辑主要由游戏控制模块实现，通过计分寄存器记录得分情况，并在游戏过程中实时显示得分。

命中状态`hitSuccess`

```
parameter
Success = 2'b10,
noneSense = 2'b00,
hitLost = 2'b01;
```

本项目的计分规则如下：

Level积分模式

- `Success`击中地鼠：加1分
- `hitLost`或者`noneSense`未击中地鼠：不得分
- 游戏结束：回合时间结束`timeIsup`，显示累计得分，如果回合内得分未达到要求得分，游戏失败，得分闪烁。

Dead死亡模式

- `Success`击中地鼠：加1分
- `hitLost`未击中地鼠：游戏直接失败，进入得分界面
- 游戏结束：计时结束`timeIsup`，显示当前回合的累计得分，如果中途漏掉地鼠，游戏失败

二进制到十进制输出

`bin2bcd`模块实现了这一转化过程，并输出到显示模块。

`score`本身为二进制计数，所以在外接了一个`binary_bcd`模块来实现进制转化到合适的输出上，十进制分数的位数是三位，每`4bit`表示一个十进制的数位。

按键交互

```
input [3:0] controlkey
```

`controlkey[0]`，`controlkey[1]`，`controlkey[2]`，`controlkey[3]`对应开发板`key1`，`key2`，`key3`，`key4`，因为有扩展键盘，这四个按键多了出来，为其赋予了额外的职能。

`key1`：游戏状态机跳转的按钮

`key2`：游戏等级增加

`key3`：游戏等级降低

`key4`：游戏模式切换

地鼠控制接口逻辑

模块接口

```
// 系统时钟1/1000频率
module mouseInterface (
    clk,
    rst,
    state,
    level,
    hit_Index,
    hitSuccess,
    mouse
);
```

模块实现了地鼠的随机生成，比对打击位置和当前地鼠的位置以输出`hitSuccess`信号，模块的书写逻辑可以非常轻松地拓展出多个地鼠共同出现的逻辑。

`input [3:0] level`：游戏难度等级

`input [3:0] state`：游戏状态

`input [15:0] hit_Index`：打击位置，一共16个，有效为1

`output reg [15:0] mouse`：输出老鼠位置，一共16个

`output reg [1:0] hitSuccess`：输出是否打击成功

随机数的发生逻辑

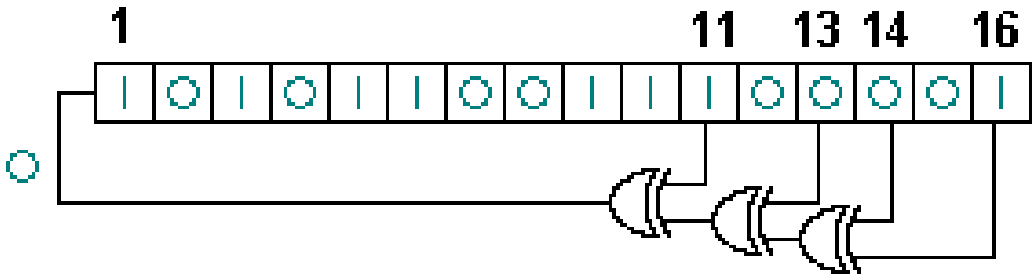
模块接口

```
// 系统时钟1/1000频率
module random (
    gamestart,
    refreshSig,
    rst,
    randout
);
```

随机数发生的逻辑只在产生地鼠的接口中定义了这样更好地实现了项目的模块分割，不同的模块之间更加独立。`gamestart`信号会让模块根据当前的运行的时间定下当前使用的随机数种子以做到“真正”的随机。

随机数生成实现

模块定义了一个`32`位的输出，中间用到了`Liner Shift Feedback Register`的结构。这个结构能够根据非`32'b0`的种子均匀地输出一个伪随机的序列。



地鼠的出现逻辑

游戏等级机制

每轮游戏有自己的等级，设定中只根据`level`对地鼠的生存周期与出现频率进行了一定的调整，并不一定合理。地属的生存时间与出现的频率是与等级线性相关的，可以在程序中很容易做到调整，在测试中，最高的等级在一分钟可以出现接近20只老鼠，最低的等级一分钟大概出现10只左右。`level`会直接影响不同模式游戏的难度。

```
reg [31:0] maxComeup;
reg [31:0] minComeup;
```

这两个变量会根据当前的等级更新自己，用来改变游戏的难度。

地鼠准备时间

```
reg [31:0] nextMousecounter;
reg [3:0] nextIndex;
reg [4:0] CurrentMouse;
reg [31:0] counter;
reg [31:0] liveTime;
```

这五个变量是与地鼠的生成相关的时间具体的逻辑不做展开介绍，总之他们实现了快速更新地鼠的生存时间以及当前地鼠的编号。而且能够将`CurrentMouse`和`counter`扩展成更高的维度，快速实现多只老鼠同时出现的逻辑。

打地鼠逻辑

打地鼠的逻辑依托于记录的`CurrentMouse`寄存器，只需要比对每一只`CurrentMouse`的`CurrentMouse[4]`是否有效，以及打击位置`hit_Index[CurrentMouse[3:0]]`是否为`1'b1`即可。不需要对所有的位置的老鼠进行比对。如果击中老鼠或者老鼠的生存周期结束，都会直接将`CurrentMouse[4]`置为`1'b0`，表示当前位置老鼠已经无效。对应的输出接口`mouse[CurrentMouse[3:0]]`也会被置为0，表示熄灭。这样一种逻辑的运行速度是很快的。打个比方就是相当于从遍历，变成了哈希。

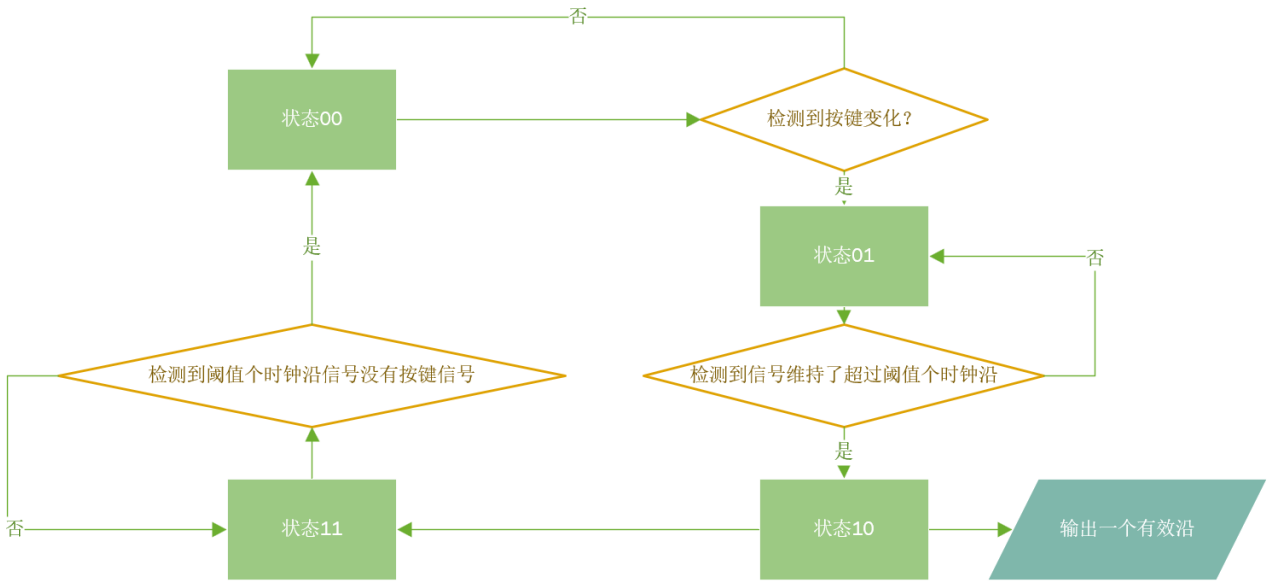
按键消抖逻辑

模块接口

```
// 系统时钟1/1000频率
module debounce (
    clk,
    button_in,
    button_out
);
```

本项目的按键消抖使用了一个简单的状态机来实现，记录按键的稳定状态，然后输出一个有效沿的信号。避免按键信号被重复输出。依靠这样的方式实现了非常好的按键消抖效果。避免了信号被反复触发带来的错误。这个状态机实现了自动的置位。非常好用。

按键消抖状态机



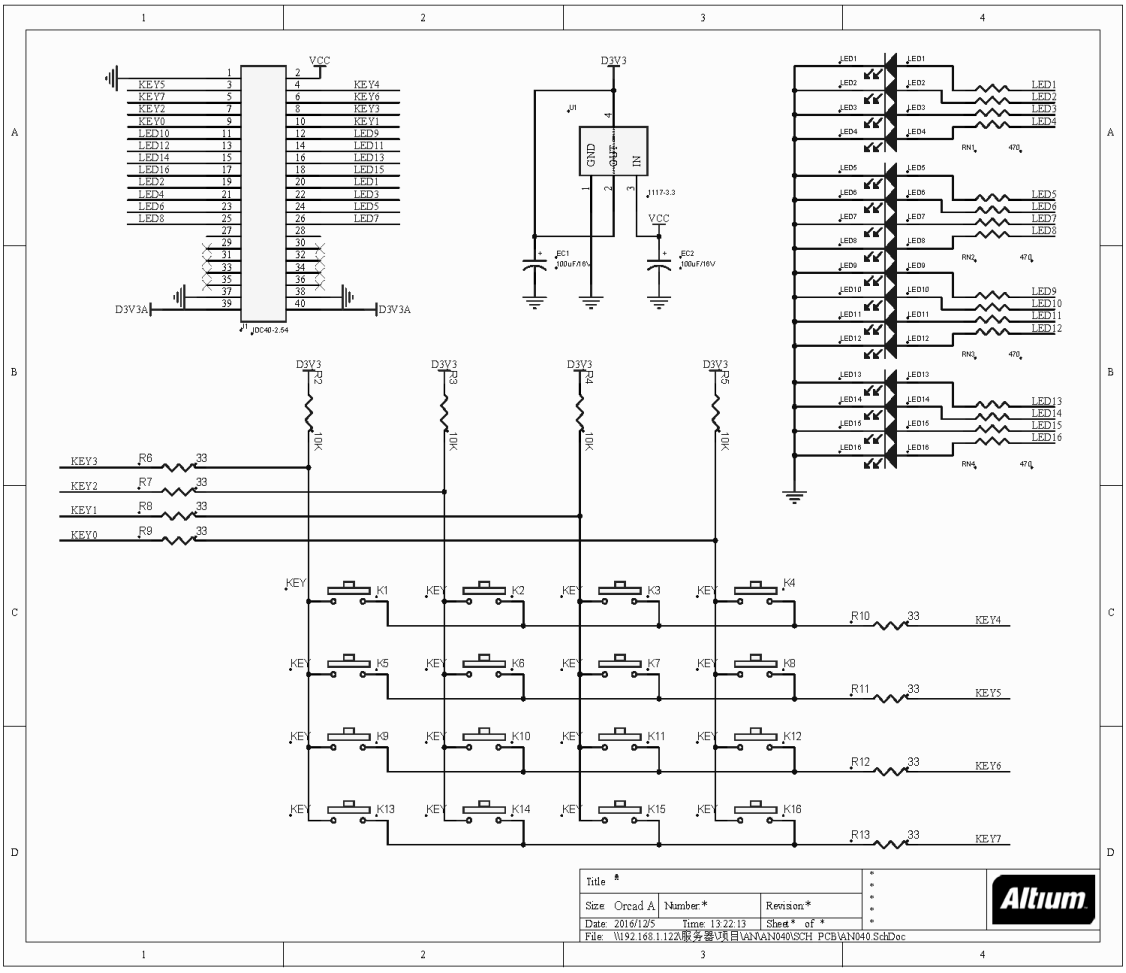
按键阵列扫描逻辑

模块接口

```
module keyboard_scan (
    clk,  // 开发板上输入时钟: 50Mhz
    clk_1000, // 输出信号维持的周期
    rst,  // 开发板上复位按键
    key_in_y,  // 输入矩阵键盘的列信号(KEY0~KEY3)
    key_out_x,  // 输出矩阵键盘的行信号(KEY4~KEY7)
    hit_Index  // 对应的输出编号
);  //键盘扫描
```

扩展键盘大小为4×4，通过4个输入与4个输出进行扫描与按键的定位。以`20ms`的周期对键盘进行反复扫描以得到一个`hit_Index`，然后输出并维持一个有效的时钟沿。

扩展模块电路示意图：



数码管扫描逻辑

数码管每`1ms`扫描给一个输出到显示端，利用人的视觉暂留实现了显示功能。利用模块化的编程方法，将这个扫描以及输出的逻辑写得非常清晰，可扩展性非常强大。

数码管输出接口

```
module smg_interface (
    input         clk,
    input         rst,
    input  [23:0] Number_Sig,
    output [ 7:0] SMG_Data,
    output [ 5:0] Scan_Sig
);
```

数码管输出得接口定义如上，其接收24位的`Number_Sig`，然后将这个信号转化成6个数码管分别的输出。每`4bit`对应一个输出位置。

数码管输出控制模块

```
module smg_control_module (
    input         clk,
    input  [23:0] Number_Sig,
    input  [ 5:0] cur_state,
    output [ 3:0] Number_Data
);
```

通过当前的状态输出一个需要显示数据到`4bit`编码上。

数码管输出扫描模块

```
module smg_scan_module (
    input        clk,
    input  [5:0] cur_state,
    output [5:0] Scan_Sig
);
```

根据当前的状态，选择数码管的选通信号。

数码管输出编码模块

```
module smg_encode_module (
    input        clk,
    input  [3:0] Number_Data,
    output [7:0] SMG_Data
);
```

将输入的`4bit`的编码输出到对应的数码管显示。

实验结果

模块波形仿真

由于篇幅限制，这里只展示三个项目测试文件的结果。

测试环境

1. `vivado version 2019.2`
2. 平台`Windows 11`

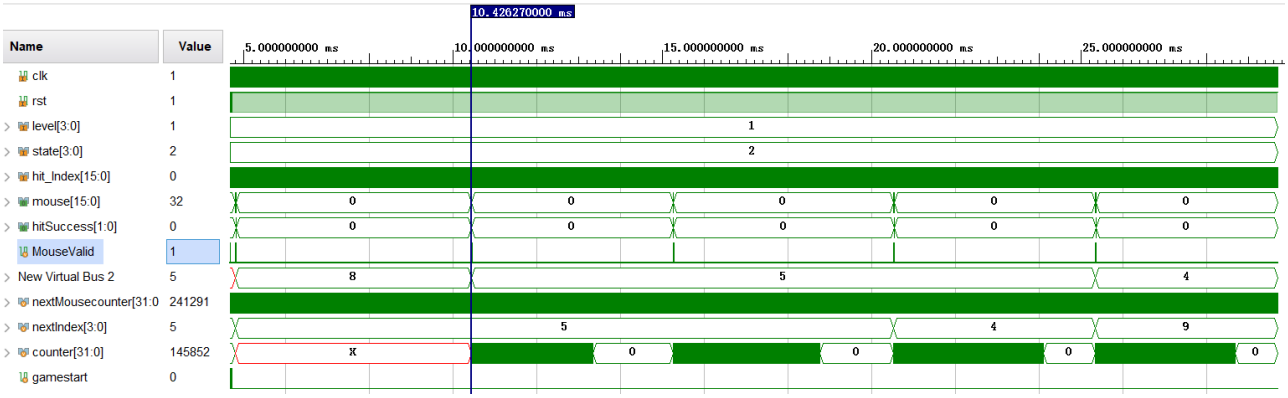
测试模块 1 `mouseInterface`

测试文件 `testmouse.v`

```
module mouseInterface_test;

  // 导入被测模块
  mouseInterface dut (
    .clk(clk),
    .rst(rst),
    .state(state),
    .level(level),
    .hit_Index(hit_Index),
    .hitSuccess(hitSuccess),
```

```
    .mouse(mouse)
  );
reg [3:0] state;
reg [3:0] level;
reg [15:0] hit_Index;
  // 定义时钟
  reg clk;
  always #10 clk = ~clk;

  // 定义复位信号
  reg rst;

  // 定义测试向量
  initial begin
    // 等待复位完成
    #1 clk = 0;
    rst = 1'b1;
    #10 rst = 1'b0;
    #10  rst = 1'b1;
    state <= 4'b0001;
    hit_Index <= 16'h0000;
    #10;
    level <= 1;
    state <= 4'b0010;

    // 等待测试完成
    end
always begin
    // 进入游戏状态
    #10000;
    hit_Index <= 16'b1111_1111_1111_1111;
    #20
    hit_Index <= 16'b0000_0000_0000_0000;
end
endmodule
```

波形结果



在这个模块中我将`hit_Index`高频地置为全1，表示打击位置全部有效，可以看到`hit_Success`在每过一段时间会出现一个有效的表示击中的上升沿。在第一只老鼠出现以前`counter`都没有任何赋值，等到`nextMousecounter`为0 才表示可以出现第一只老鼠。另外也可以看到，不同的老鼠的生存周期并不相同，出现的时间也无法预测。符合要求。

测试模块 2 `random`

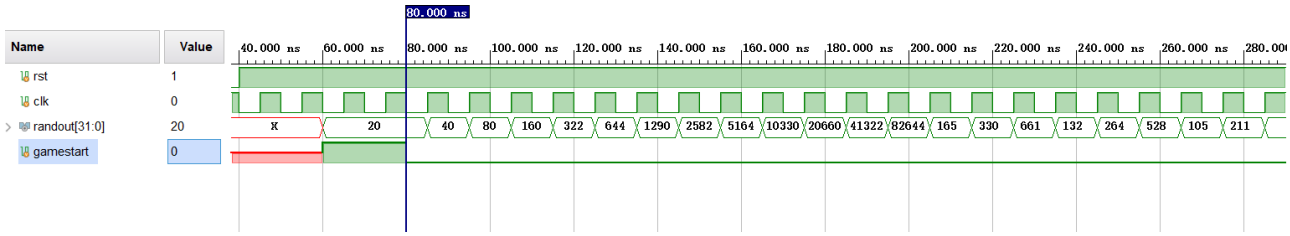
测试文件 `test_random.v`

```
module testrandom;
  // Inputs
  reg rst;
  reg clk = 0;
  // Outputs
  wire [31:0] randout;
    reg gamestart;
  // Instantiate the module to be tested
  random dut (
  .gamestart(gamestart),
    .refreshSig(clk),
    .rst(rst),
    .randout(randout)
  );
  // Clock generation
  always #5 clk = ~clk;
  // Testbench
  initial begin
    // Reset the DUT
    rst = 1;
    #20;
    rst = 0;
    #20
```

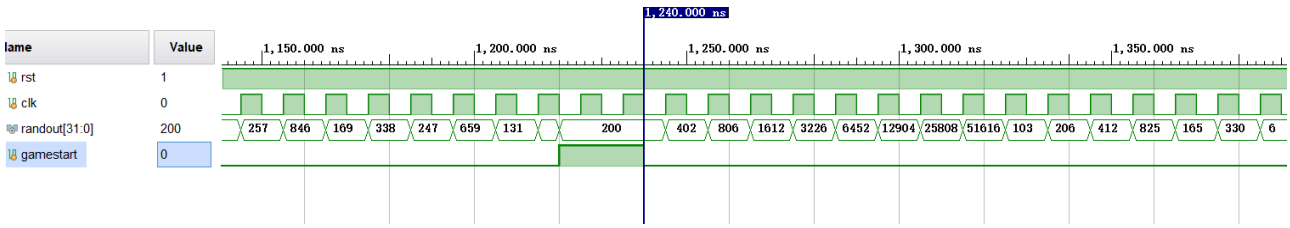
```
24      rst = 1;
25      #20
26      gamestart <= 1;
27      #20
28      gamestart <= 0;
29      #200
30      rst = 1;
31      #20;
32      rst = 0;
33      #20
34      rst = 1;
35      #900
36      gamestart <= 1;
37      #20
38      gamestart <= 0;
39      end
40      // End the simulation
41  endmodule
```

波形结果

第一次重置随机数：



第二次重置随机数：



可以看到`rst`信号之后经过不同的时间，产生的信号也有差别，可以用这种方式实现近似的真随机效果。

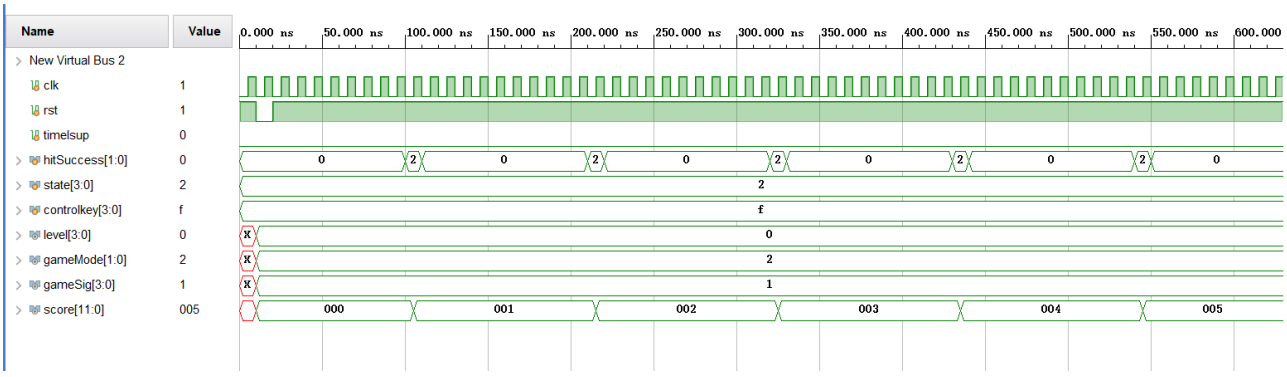
测试模块 3 `gameControl`

测试文件 `tb_gameControl.v`

```
`timescale 1ns/1ns

module testbench;

  // 定义模块接口
  reg clk, rst, timeIsup;
  reg [1:0] hitSuccess;
  reg [3:0] state;
  reg [3:0] controlkey;
  wire [3:0] level;
  wire [1:0] gameMode;
  wire [3:0] gameSig;
  wire [11:0] score;

  // 实例化被测试的模块
  gameControl dut(
    .clk(clk),
    .rst(rst),
    .timeIsup(timeIsup),
    .hitSuccess(hitSuccess),
    .state(state),
    .controlkey(controlkey),
    .level(level),
    .gameMode(gameMode),
    .gameSig(gameSig),
    .score(score)
  );

  // 定义状态机常数
  parameter beforeGame = 4'b0001, inGame = 4'b0010, GameLost = 4'b0100, GameWin = 4'b1000;
  // 定义信号常数
  parameter keepCurrent = 4'b0001, game_win = 4'b0010, start_press = 4'b0100, game_lost = 4'b1000;
  // 游戏模式常数
  parameter Level = 2'b10, Dead = 2'b01;
  // hit状态
  parameter Success = 2'b10, noneSense = 2'b00, hitLost = 2'b01;
  // 初始化输入信号
```

```
38    initial begin
39      clk = 0;
40      rst = 1;
41      timeIsup = 0;
42      hitSuccess = noneSense;
43      state = inGame;
44      controlkey = 4'b1111;
45      #10 rst = 0;
46      #10 rst = 1;
47    end
48
49    // 时钟生成器
50    always #5 clk = ~clk;
51    // 仿真测试程序
52    initial begin
53      // 检查重置信号是否正常工作
54      state = inGame;
55      // 测试结束
56      #1000000 timeIsup = 1;
57    end
58  always begin
59      #100 hitSuccess = Success;
60      #10 hitSuccess = noneSense;
61  end
62  endmodule
```

波形结果



这是游戏数据的控制模块的波形，每接收一个有效的击中信号，会让积分增加。符合预期。

综合结果显示

总的来说，通过优化与整合，项目的结果表现良好，时钟的频率可以继续往下面降低以降低功耗，实际的功耗可以做到更低。

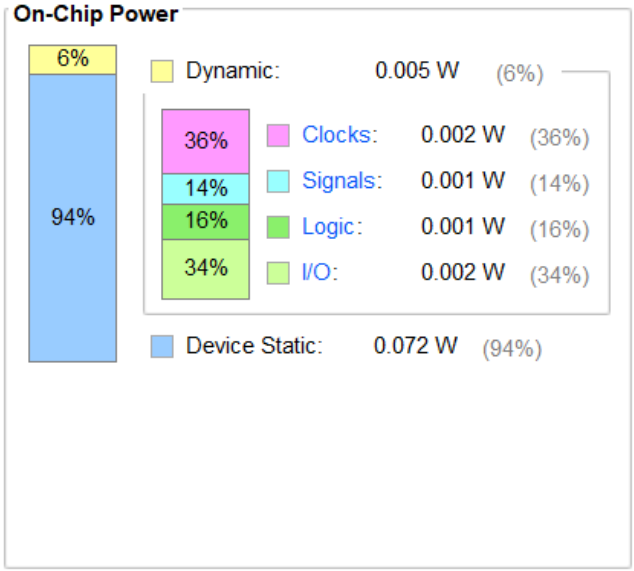
power

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.076 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**25.2°C**
Thermal Margin:	59.8°C (21.2 W)
Effective ϑJA:	2.8°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Timing

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	13.730 ns	Worst Hold Slack (WHS):	0.172 ns	Worst Pulse Width Slack (WPWS):	9.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	508	Total Number of Endpoints:	508	Total Number of Endpoints:	257

All user specified timing constraints are met.

实验结果展示

视频链接：

https://pan.baidu.com/s/1k-poVcG3wJccFlI-gpetEw
提取码:b921

问题和解决方案

系统设计中遇到的问题

在本项目的实现过程中，我遇到了一些问题，如按键抖动、游戏状态转移等问题。通过调试和优化。

解决方案及效果评估

1. 按键抖动问题：通过增加按键消抖电路来解决，消抖时间设置为1000倍系统时钟周期。效果评估：按键消抖后，按键响应更加稳定，游戏体验更好。
2. 游戏状态转移问题：通过添加状态机，根据当前状态进行状态转移，解决了游戏状态转移不清晰的问题。效果评估：游戏状态转移更加清晰，游戏逻辑更加流畅。
3. 显示错误问题：通过对时序进行优化，保证了时序的准确性，解决了显示错误问题。效果评估：显示准确无误，游戏体验更加良好。
4. 时序错位问题：通过对时钟信号进行同步，解决了时序错位问题。效果评估：时序稳定，游戏体验更加流畅。

总结

项目总结

本项目旨在设计一个基于`FPGA`技术的打地鼠游戏，通过该项目的开发和实现，进一步了解了`FPGA`硬件开发的流程和方法。在项目中，开发者成功地实现了地鼠的出现和消失，玩家的按键输入以及游戏得分等功能。虽然在开发过程中遇到了一些问题，如按键抖动、游戏状态转移、显示错误、时序错位等问题，但是通过调试和优化，这些问题都得到了有效的解决。

在项目开发过程中，虽然开发者是第一次使用`FPGA`硬件进行实际的实现，但是在项目中留下了很多可以扩展的接口，可以方便地扩展其他的硬件设备并实现更多的功能。同时，开发者也深刻地认识到了硬件开发过程中的规范性和细节性对于项目的重要性，这为以后的硬件项目开发积累了经验和教训。

总之，本项目的开发和实现不仅提高了开发者的硬件开发水平，也为开发者继续学习FPGA硬件开发提供了一个窗口。

后续改进的思路

在本项目的开发和实现过程中，虽然已经成功地实现了地鼠的出现和消失、玩家的按键输入、游戏得分等功能，但是还有一些可以进一步改进和优化的地方。以下是一些后续改进的思路：

1. 细化难度等级：目前游戏难度等级并不太合理，可以考虑合理化不同的难度等级，对地鼠出现和消失的速度、数量等进行测试增加，以增加游戏的挑战性和可玩性。
2. 增加音效反馈：目前游戏没有音效反馈，可以考虑增加击中地鼠的音效、游戏得分的音效等，以提高游戏的趣味性和交互性。但这点主要受限于硬件，实际上没有实现难度。
3. 优化游戏显示效果：可以考虑优化游戏的显示效果，例如使用外接显示器、增加背景图片、增加地鼠出现和消失的动画效果等，以提高游戏的美观性和趣味性。
4. 扩展其他硬件设备：开发者在项目中留下了很多可以扩展接口的结构，可以考虑扩展其他硬件设备，例如添加外部音箱等，以增加游戏的交互性和娱乐性。

总之，以上是一些后续改进的思路，通过不断地优化和改进，可以让这款基于FPGA技术的打地鼠游戏更加有趣、有挑战性、有多样性，更符合现代玩家的需求。