



# 《数字逻辑设计》课程设计报告

## 基于 FPGA 的打地鼠游戏

组 长： XX

学 号： 324010XXXX

组 员： XXX

学 号： 324010XXXX

最后编辑日期： 2025 年 6 月 21 日

# 目录

1 项目概述 .....	3
1.1. 简介 .....	3
1.2. 使用说明 .....	3
1.3. 硬件平台 .....	5
1.4. 开发环境与工具链 .....	5
2 项目实现 .....	6
2.1. 项目架构 .....	6
2.2. 外设驱动 .....	6
2.2.1. VGA 显示 .....	6
2.2.2. PS/2 鼠标 .....	7
2.2.3. 七段数码管 .....	7
2.2.4. 蜂鸣器 .....	8
2.3. 游戏主逻辑 .....	8
2.4. 音视频管理 .....	9
2.4.1. 声音管理 .....	9
2.4.2. 显示管理 .....	9
2.5. 其他模块 .....	11
2.5.1. 随机数生成器 .....	11
2.5.2. 时钟分频模块 .....	11
2.5.3. 外设测试模块 .....	11
2.6. 模块仿真 .....	12
2.6.1. 随机数生成器 .....	12
2.6.2. P2S 七段数码管串转并模块 .....	12
3 总结与反馈 .....	13
3.1. 项目总结与参考 .....	13
3.2. 开发日志与分工贡献 .....	13
3.3. 一些心得 .....	14
3.4. 后记 .....	14

# 项目概述

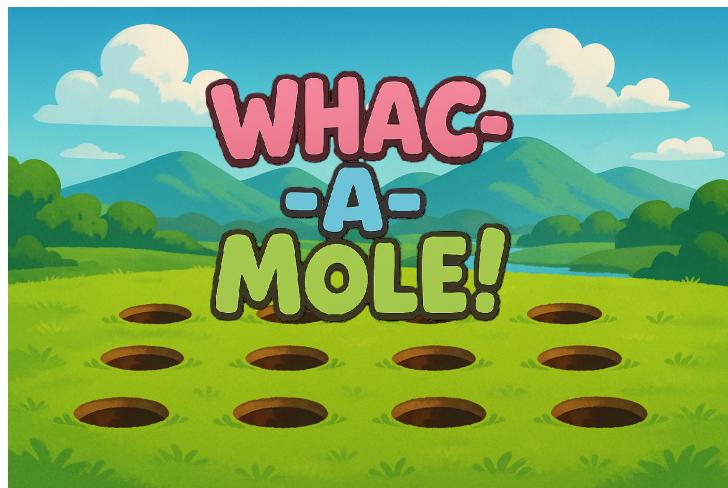
## 1.1. 简介

本系统是基于FPGA平台实现的“打地鼠”电子游戏。玩家通过鼠标与屏幕中的地鼠互动，配合VGA显示器显示界面，复现街机游戏中经典的打地鼠场景。玩家根据地鼠出现的位置迅速作出反应进行打击。

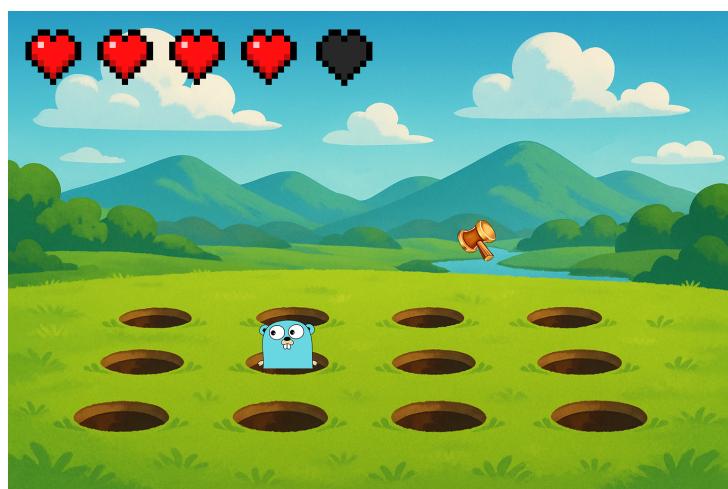
## 1.2. 使用说明

将比特流文件烧录到FPGA开发板上，连接好PS/2鼠标和VGA显示器。

显示器上会展示启动页面<sup>1</sup>:



单击鼠标左键开始游戏，游戏界面如下：



---

<sup>1</sup>由于无法直接对VGA屏幕截图，下面的图片为后期合成，仅供效果演示

玩家可以通过鼠标移动锤子的指针位置，当地鼠出现时，点击鼠标左键进行打击。每次成功打击地鼠会增加分数，击打时间越早分数越高，并播放相应的音效。游戏会随机生成地鼠出现的位置和时间。

玩家分数越高，地鼠出现的频率和速度会逐渐增加，游戏难度也会随之提升。分数足够时，游戏胜利结束，显示胜利画面：



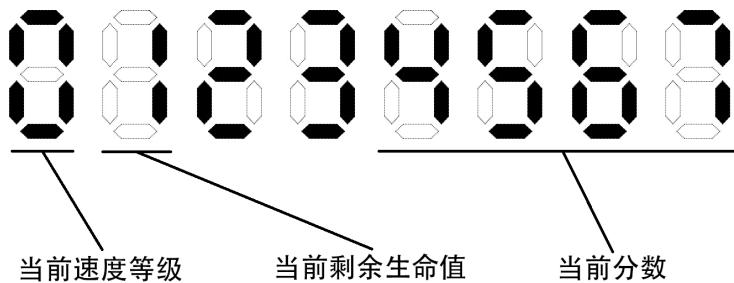
当玩家漏掉地鼠或未能及时打击，会扣除一条生命值。游戏开始时玩家有五条生命值，生命值耗尽后游戏结束，显示失败画面：



游戏胜利或失败结束后，玩家可以点击鼠标右键返回到启动页面，重新开始游戏。

除 VGA 屏幕外，本项目还使用了七段数码管显示游戏状态信息，包括当前速度等级、分數和剩余生命值等。数码管会实时更新。

具体信息如下：



### 1.3. 硬件平台

本项目基于浙江大学东四教学楼 509 教室的 SWORD 4.0 平台开发，同时使用了 PS/2 鼠标、VGA 显示器等外设。

在本项目中，我们利用 SWORD 4.0 平台上集成的 Xilinx Kintex™-7 FPGA 作为核心处理单元，通过其丰富的 I/O 接口连接并控制外部设备。我们使用了 PS/2 鼠标作为用户输入设备，可接收用户的实时控制指令；VGA 显示器用于图形界面的输出展示。

### 1.4. 开发环境与工具链

本项目采用 **Vivado 2024.2** 作为主要开发平台，所有模块均使用 **Verilog HDL** 编写。

图片资源的转换处理使用了 Python **Pillow** 库，生成适合 FPGA 处理的 coe 格式。

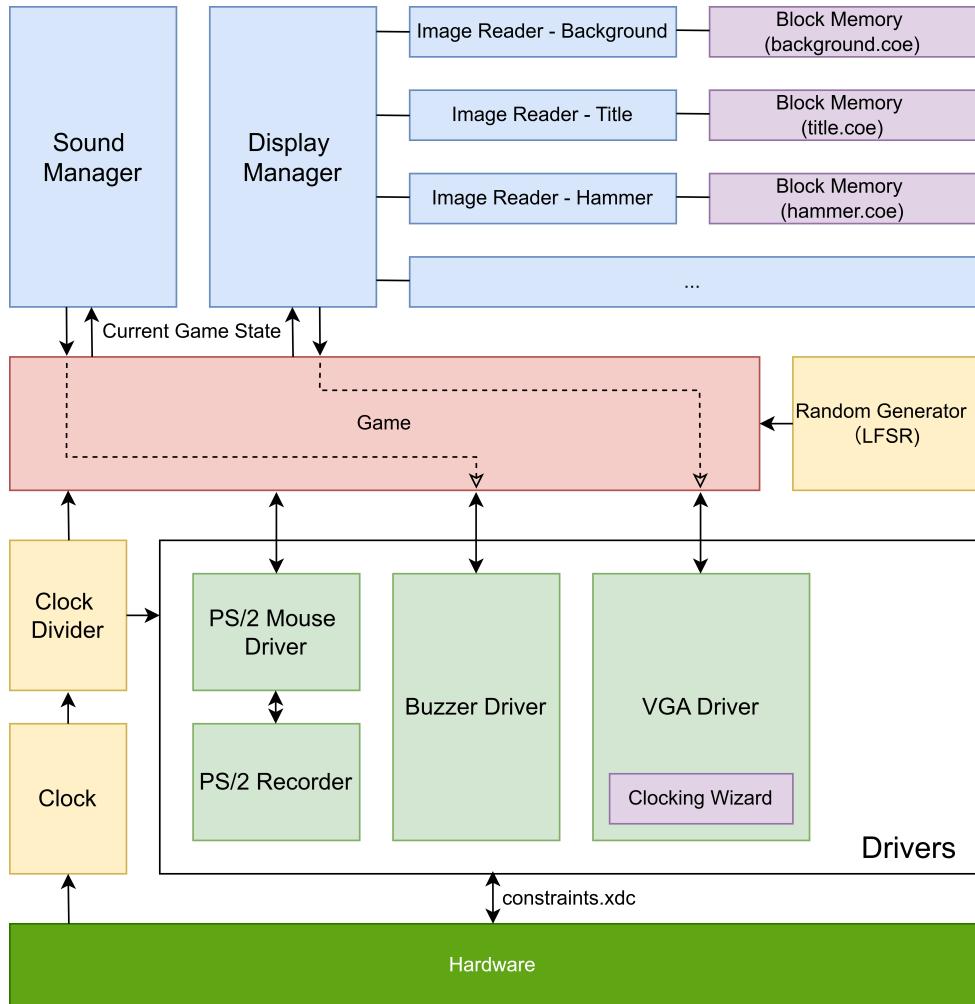
项目版本管理使用 **Git** 进行，代码托管在 [该项目的 GitHub 仓库](#) 中。

报告使用 **VSCode+Tinymist+Typst** 编写，报告中的结构图和流程图使用 **draw.io** 绘制。

# 项目实现

## 2.1. 项目架构

本项目的架构如下图所示：



项目主要分为以下几个模块：外设驱动、游戏主逻辑和音视频管理，详细信息将在下面介绍。

## 2.2. 外设驱动

### 2.2.1. VGA 显示

VGA 显示协议通过模拟信号配合同步信号来驱动显示器逐行扫描显示图像。图像数据以 RGB 信号传输，通过水平同步信号 hsync 和垂直同步信号 vsync 控制扫描时序。显示器按照从左到右、从上到下的顺序进行逐行扫描。整个过程中，图像内容一般存储在帧缓存中，按照 VGA 规定的分辨率和刷新频率，周期性地输出到显示设备上。

权衡图片资源的大小和 SWORD 板的 Block Memory 容量后，我将游戏界面分辨率设置为 640x480。本项目对实验文档中的 vgac 模块进行了简单改写作为 VGA 驱动，显示内容主要由后续的显示管理模块 Display Manager 进行控制。

VGA 时钟由 Clocking Wizard IP 核生成，时钟频率为 25.175MHz。

本模块的实现主要位于 `vga/vga_driver.v` 文件中。

### 2.2.2. PS/2 鼠标

为了实现鼠标输入，我对实验室中的不同鼠标进行了大量尝试，发现不同鼠标经过开发板转换后的 PS/2 信号存在差异。最终找到了一个较为老旧的与 PS/2 鼠标标准协议较为接近的可用鼠标。

鼠标每次移动或操作时会连续发送三个字节的 PS/2 数据包：

字节	位	PS/2 鼠标标准	实验室鼠标
1	0	左键状态；1 = 按下	左键状态；1 = 按下
1	1	右键状态；1 = 按下	右键状态；1 = 按下
1	2	中键状态；1 = 按下	恒为 0
1	3	保留	恒为 0
1	4	X 方向符号位；1 = 负	恒为 0
1	5	Y 方向符号位；1 = 负	恒为 0
1	6	保留	恒为 0
1	7	保留	恒为 0
2	0-7	X 方向位移量	X 方向位移量（有符号整数）
3	0-7	Y 方向位移量	Y 方向位移量（有符号整数）

为了实现 PS/2 鼠标数据包的接收和解析，我编写了 PS2 Recorder 模块，该模块实际上是一个 PS/2 数据包上层的串进并出转换器。它接收三个 PS/2 信号后发出激活信号，使上层的 PS2 Mouse Driver 解析当前三个字节的数据包。

PS2 Mouse Driver 模块会根据 PS/2 鼠标协议解析数据包，它使用一个累加器将鼠标的位置信息转换为游戏界面坐标系下的像素位置。同时处理鼠标按键状态，供游戏逻辑模块使用。

该模块的实现主要位于 `mouse/ps2_recorder.v` 和 `mouse/ps2_mouse_driver.v` 文件中。

### 2.2.3. 七段数码管

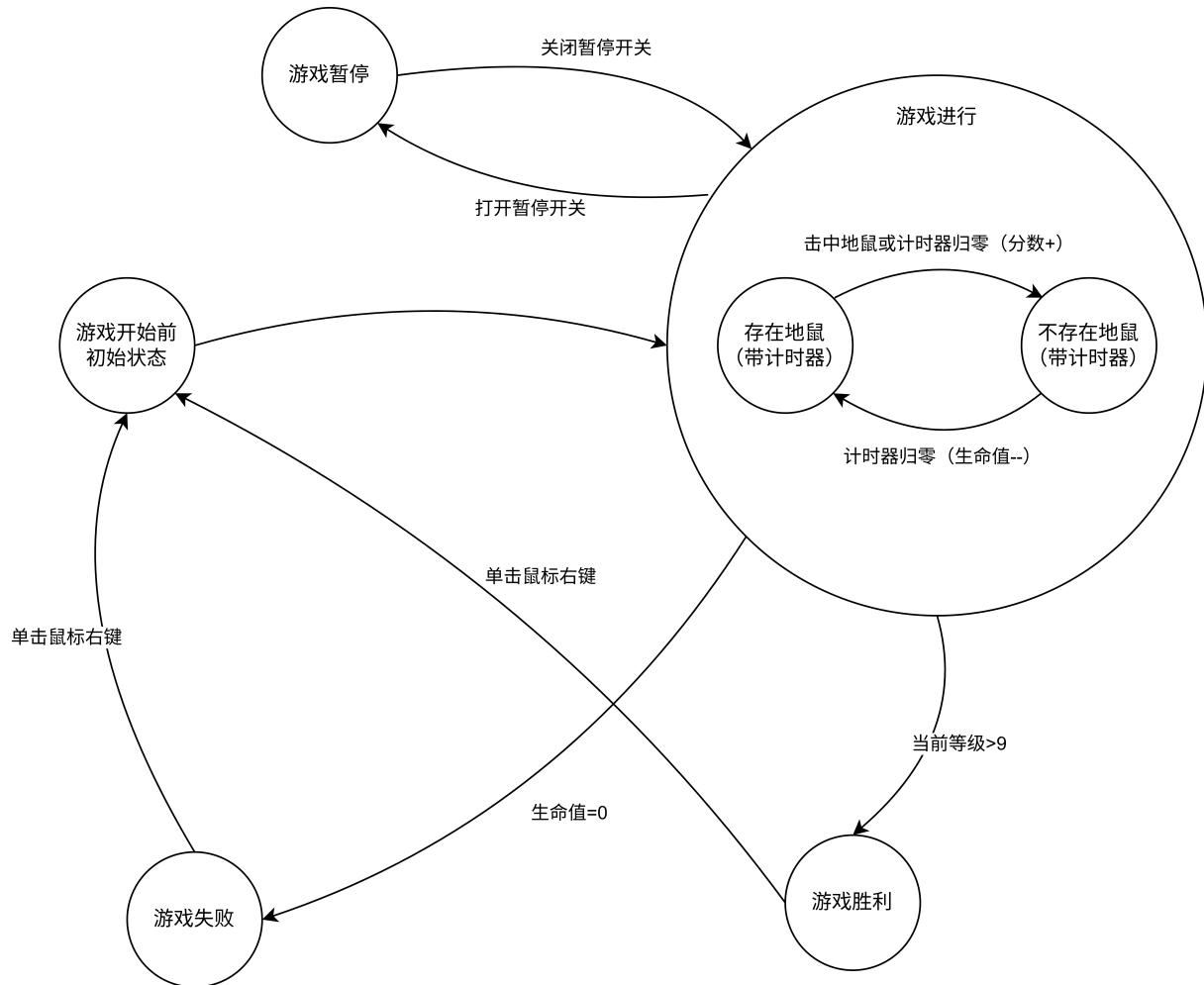
七段数码管的驱动采用了 Lab D - Shift Register 中的七段数码管驱动模块，实际上就是对 P2S 模块的简单封装。

#### 2.2.4. 蜂鸣器

蜂鸣器模块本身较为简单，蜂鸣器驱动模块只对 64 种常见音频频率进行了预设。具体音频播放逻辑由声音管理模块 Sound Manager 进行控制。

### 2.3. 游戏主逻辑

游戏的主逻辑是一个有限状态机，为了便于表现，这里对它进行了高度简化，可以表示为下面的不严格的状态转移图：



由于具体的状态转移图较为复杂，这里只给出一个简化的版本。实际的状态转移图包含了更多细节，例如地鼠出现的具体位置、分数计算、生命值管理等。

该模块的实现主要位于 `game/game.v` 文件中。

## 2.4. 音视频管理

### 2.4.1. 声音管理

声音管理模块 Sound Manager (Buzzer Manager) 负责处理游戏中的音效播放。它使用了两个简单的栈来存储正在播放的音频，并通过游戏状态的变化来触发不同的声音播放事件。

声音模块的主要状态有 正在播放、未在播放：

当状态为未在播放时，声音管理模块在每个时钟周期会检查游戏状态与上一个状态是否发生变化，如果发生需要播放音频的变化（包括鼠标点击、分数上升、损失生命、游戏胜利等），则根据改变化加载对应的音频到音符栈中，并转移到正在播放状态。

当状态为正在播放时，声音管理模块根据当前音符栈调整音频输出的频率，并在音符播放完毕后转移回未在播放状态。

#### 音频的存储与播放

声音管理模块将音频存储为 register 数组上的两个栈，分别存储音符的序号和对应的播放时间。每个音符的序号和持续时间都存储在这两个栈中。

当状态为正在播放时，音频的播放通过一个计数器来控制，当计数器达到当前栈顶音符的持续时间时，将音符出栈，并将计数器重置为 0。然后根据下一个音符的序号设置音频输出的频率。

该模块的实现主要位于 `game/buzzer_controller.v` 文件中。

### 2.4.2. 显示管理

显示管理模块 Display Manager 直接控制 VGA 驱动，根据游戏状态和鼠标位置来更新显示内容。

图片资源的存储使用了 coe 格式，存储在项目外的 `images` 目录下。每个图片资源都对应一个 coe 文件，文件中存储了图片的像素数据。除了背景图为 RGB 格式外，其他的图片均为 RGBA 格式以实现透明效果。

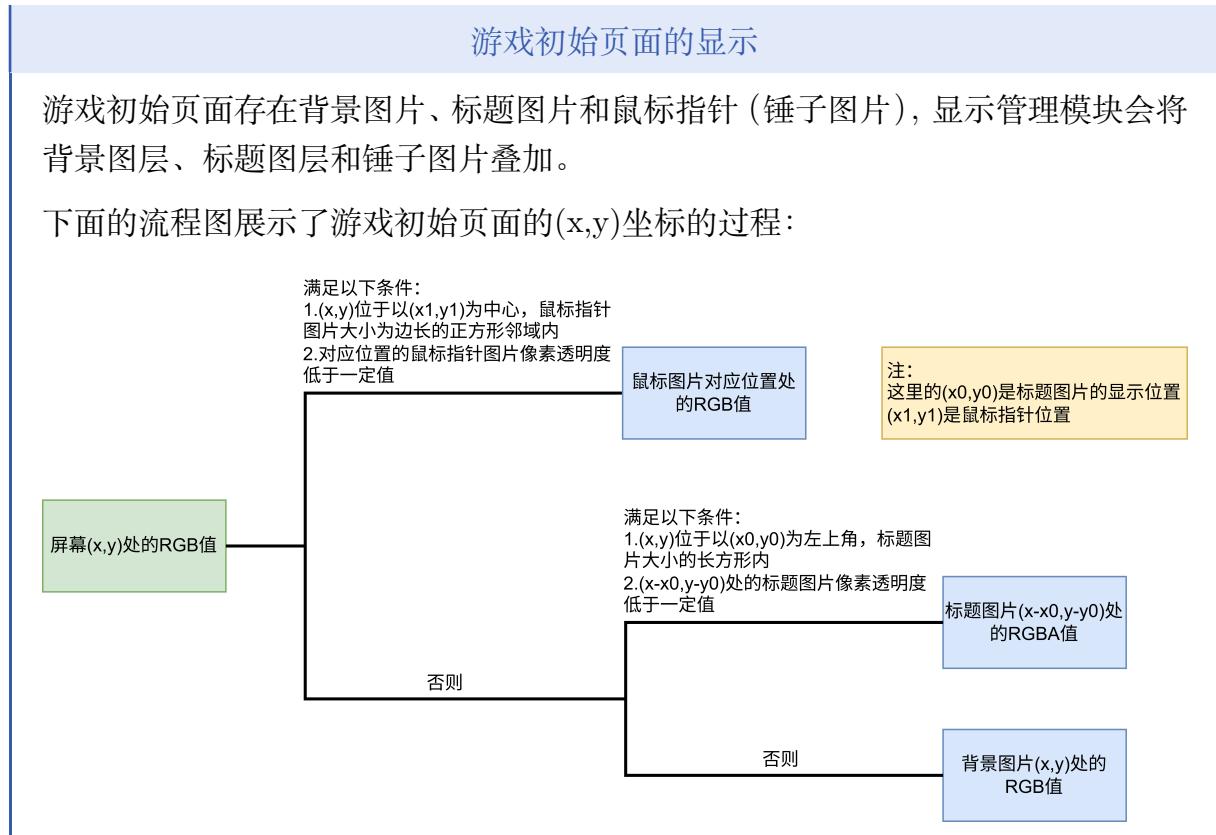
每一个图片都封装了一个 Image Reader 模块，该模块负责从 coe 文件初始化的 Block Memory 中读取图片的像素数据。

显示管理模块根据游戏状态来确定显示内容，当需要确定屏幕上某一个像素点的颜色时，显示管理模块会根据当前的游戏状态和鼠标位置来选择对应的图片资源，并从 coe 文件中读取对应的像素数据。

#### 2.4.2.1. 图片的叠加

像素颜色的确定实际上是显示图层的串行叠加过程。每个图层都有一个对应的图片资源，当多个图层重叠时，显示管理模块会根据透明度和游戏状态来决定是否对当前像素进行覆盖。

下面举一个简单的例子：



实际游戏中，存在大量的图层叠加和透明度处理。显示管理模块会根据游戏状态和鼠标位置来动态调整显示内容。

由于本项目没有半透明的图层，这里只处理全透明和不透明的情况。实际上半透明的图层处理也并不困难，按照透明度计算 RGB 每个通道数值的乘积之和，在取变量高位作为等效除法即可。相当于：

$$R_{\text{mixed}} = \frac{R_{\text{current}} \times (16 - A_{\text{newlayer}}) + R_{\text{newlayer}} \times A_{\text{newlayer}}}{16}$$

$$G_{\text{mixed}} = \frac{G_{\text{current}} \times (16 - A_{\text{newlayer}}) + G_{\text{newlayer}} \times A_{\text{newlayer}}}{16}$$

$$B_{\text{mixed}} = \frac{B_{\text{current}} \times (16 - A_{\text{newlayer}}) + B_{\text{newlayer}} \times A_{\text{newlayer}}}{16}$$

### 2.4.2.2. 地鼠动画

地鼠的出入动画是通过记录地鼠实时高度实现的。地鼠的高度会随着时间变化而变化，形成一个简单的动画效果。

每个地鼠的高度由一个计数器控制，计数器具有上限和下限，其变化由游戏主逻辑中的地鼠是否出现这一变量决定。地鼠处于出现状态时，每个时钟周期计数器增加，计数器达到上限时不再增加；当地鼠处于消失状态时，计数器每个时钟周期减少，计数器达到下限时不再减少。

地鼠的高度变化会影响显示管理模块中地鼠图片的显示位置和显示高度，从而实现地鼠的动画效果。

该模块的实现主要位于 `game/display_manager.v` 文件中。

## 2.5. 其他模块

### 2.5.1. 随机数生成器

游戏中地鼠出现的位置和时间是随机生成的。为了实现这一功能，我使用了一个简单的伪随机数生成器。

随机生成器模块使用了一个线性反馈移位寄存器（LFSR）来生成伪随机数。LFSR 是一种常用的伪随机数生成算法，具有较好的随机性。

由于该项目中 LFSR 在每个时钟周期都会生成一个新的伪随机数，而玩家的操作时间间隔是会发生变化的，所以该项目中不会出现每次启动游戏时地鼠出现位置和时间完全相同的情况。

### 2.5.2. 时钟分频模块

时钟分频模块采用了自 [Lab 7 - Multiplexer](#) 以来一直使用的时钟分频模块。该模块通过计数器实现时钟信号的分频，生成不同频率的时钟信号供其他模块使用。

### 2.5.3. 外设测试模块

本项目使用了仿真和上板测试两种方式来验证模块的正确性。由于本项目的外设驱动模块较为复杂，所以主要采用的是上板测试的方式。

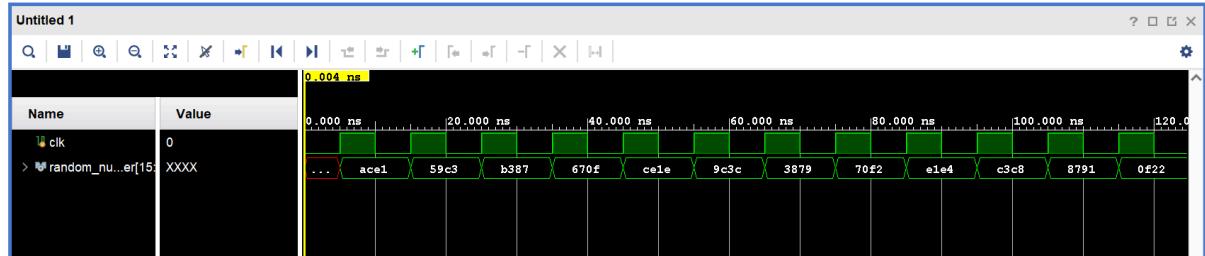
不同模块上板测试的 Top 模块和对应的约束文件都位于 `test` 目录下，包括鼠标、蜂鸣器、七段数码管、VGA 显示的测试模块以及它们的协同测试模块。

如果你正在 SWORD 上进行 FPGA 开发并在驱动方面遇到问题，可以参考这些测试模块，它们提供了对外设的最小功能实现。

## 2.6. 模块仿真

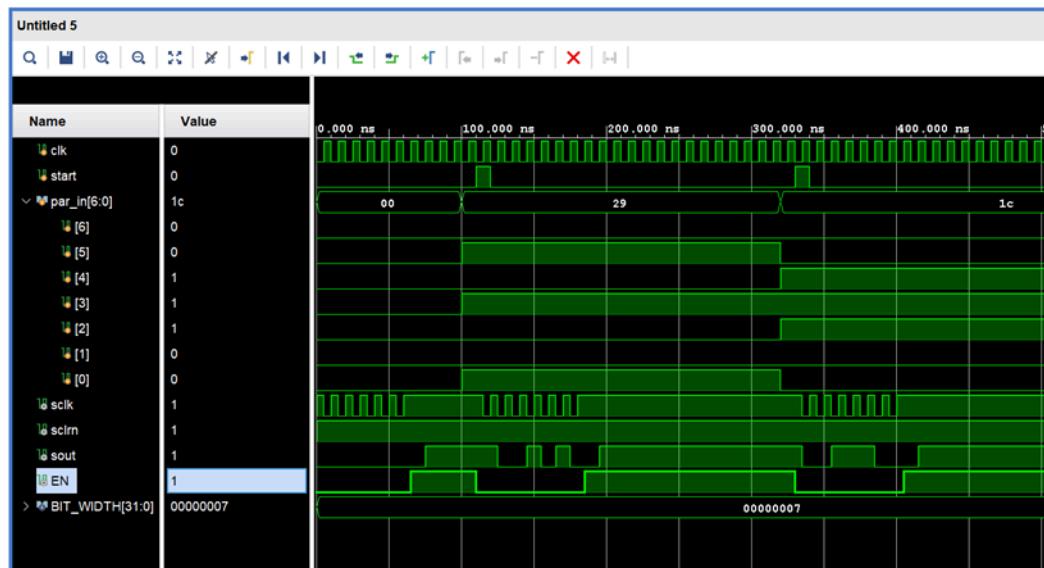
尽管主要使用的调试方法是编写独立的测试模块上板测试(见上文“外设测试模块”),该项目对两个耦合程度较低的基础模块进行了仿真测试。

### 2.6.1. 随机数生成器



正常生成伪随机，模块工作正常。

### 2.6.2. P2S 七段数码管串转并模块



正常进行串并转换，模块工作正常。

## 总结与反馈

### 3.1. 项目总结与参考

本项目利用了 SWORD 平台上的各种资源和外设，成功实现了一个基于 FPGA 的打地鼠游戏。项目功能完整，运行流畅，没有发现明显的问题。

**本项目绝大部分内容为独立编写，VGA 驱动参考了实验文档外设使用指导中的 vgac 模块，但对其进行了一定程度的重构，提高了可读性。**

项目在 Vivado 2024.2 上开发，使用的 IP 核包括 Clocking Wizard、Block Memory Generator，它们的版权归属 Xilinx 公司所有。

项目的图片资源部分由 AI 生成，部分来自网络。

其中 Golang Gopher 图片采用 CC BY 4.0 许可证发布，原作者为 Renée French。  
首页浙江大学的校徽图片的知识产权归浙江大学所有。

本报告的 Typst 模板以 MIT 协议开源于 GitHub，原作者为 [memset0](#)，仓库地址 [见此](#)。  
我对该模板做了大量修改以适应撰写本报告的需求，主要修改内容包括：重构封面、修改目录样式、修改字体。

本项目使用的 SWORD 4.0 开发板由浙江大学计算机科学与技术学院提供，是上海星灯智能科技有限公司的产品。

项目演示视频：

链接: <https://pan.baidu.com/s/1gGVmSN90zO1TJtL5yUlONw?pwd=4md8>

提取码: 4md8

### 3.2. 开发日志与分工贡献

项目完全使用 Git 进行版本管理，所有代码均托管在 GitHub 上。项目的开发日志和分工贡献可见于 GitHub 仓库的提交记录。

提交日志: <https://github.com/eWloYW8/FPGA-Whac-A-Mole/commits/master/>

贡献: <https://github.com/eWloYW8/FPGA-Whac-A-Mole/graphs/contributors>

作为组长，该项目的大部分代码和报告由我编写，一方面是因为我推进进度较快，希望在考试前尽快完成项目，同时具有较好的开发能力和对项目的整体把控。

组员同学提供了一些有益的建议和体验反馈，同时完成了部分音频的设计和项目演示视频的讲解部分，帮助我更好地完成项目。

### 3.3. 一些心得

经过穿插在考试周的几个星期的开发，本项目最终顺利完成。在东四教学楼 509 教室全天开发调试、研究不同型号的鼠标、一边复习考试一边修 Bug 的经历将成为我大学生活中难忘的回忆。

我在计算机软件方面具有一定的基础，但在硬件方面的经验较少。经过这一整个学期的数逻学习，我对计算机硬件的工作原理有了更深入的理解。上这门课的初期，我对计算机硬件的理解只是枯燥无味的电路图和逻辑门，当时的我认为软件的开发是一种自由的艺术创作，而硬件则充满着死板的规则和枯燥的重复排列单元。通过课程的 Lab 和这次课程设计，我逐渐认识到计算机硬件的魅力所在：**计算机的硬件系统也可以向软件一样进行模块化和抽象化，通过不同的模块组合自由地实现复杂的功能。**

感谢蔡铭老师在课上的耐心讲解和指导、助教们的耐心解答和瓜豪学长的精品实验文档，以及室友的经验和组员的支持，让我在这个项目中学到了很多。

作为计算机科学与技术专业的学生，我现在深刻认识到计算机硬件和软件的紧密结合是计算机科学的核心。未来我还会面临计算机组成、体系结构等更为深入的系统类课程，相信那时的我也能保持写下这篇报告时的热情和对计算机系统的好奇。

### 3.4. 后记

关于地鼠图片的选取：

本人是 **Golang** 语言的爱好者，本项目开发过程中没有一只 **Gopher** 受到伤害。