《数字逻辑设计》设计报告

马昕宇　李晨昊

2019-6-30

目录

1 总体设计思路

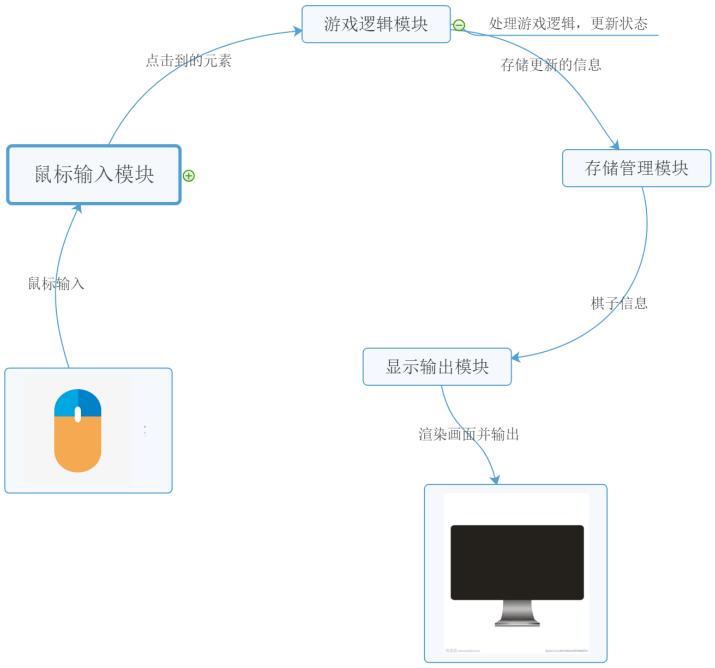
1.1 项目概述

在 FPGA 平台上设计数字逻辑系统，使用 VHDL 语言实现象棋翻棋/象棋揭棋游戏。游戏使用 PS/2 鼠标作为输入,VGA 接口连接显示器作为输出。

象棋翻棋和象棋揭棋都来源于一款手机上的象棋游戏。象棋翻棋大概的游戏流程是：游戏开局时，大多数棋子暗置在半边象棋棋盘上；红黑两方轮流下棋，每一步可以选择翻开一个暗置的棋子或者移动本方的一个棋子；棋子间按照一定的顺序可以互相吃。游戏的胜利目标为吃掉对方足够多的棋子。在此基础上可以还附加一些额外的逻辑以增加趣味性。象棋揭棋大概的游戏流程是在正常的象棋棋盘上随机摆放象棋棋子，一方除了将/帅外的棋子随机暗置在正常的象棋棋子位置上，棋子第一次被移动时翻开，棋子翻开前当作它所处的位置处的棋子使用，翻开后当作正常棋子使用。游戏的胜利目标为吃掉对方的将/帅。

1.2 模块划分

对于翻棋和揭棋各自编写了一个工程，两个工程的模块划分是类似的。系统整体划分为四个模块：鼠标输入，游戏逻辑，存储管理及显示输出。模块各自的功能和它们的交互关系如下图所示：



详细解释如下：

1. 鼠标输入模块

   包含一个底层接口和一个高层接口。底层接口负责接受并处理 PS/2 鼠标输入信号，高层接口负责接受底层接口的信号，将它转化成对于棋子的实际操作，提供给游戏逻辑模块。

2. 游戏逻辑模块

   管理棋局包含的信息，例如棋盘，棋子，棋手剩余血量等状态。它负责接受输入模块提供的信息，并且执行象棋翻棋的游戏逻辑。逻辑模块的状态发生改变后，显示输出模块会收到信号，从而可以显示出棋子移动，落子等动画效果。

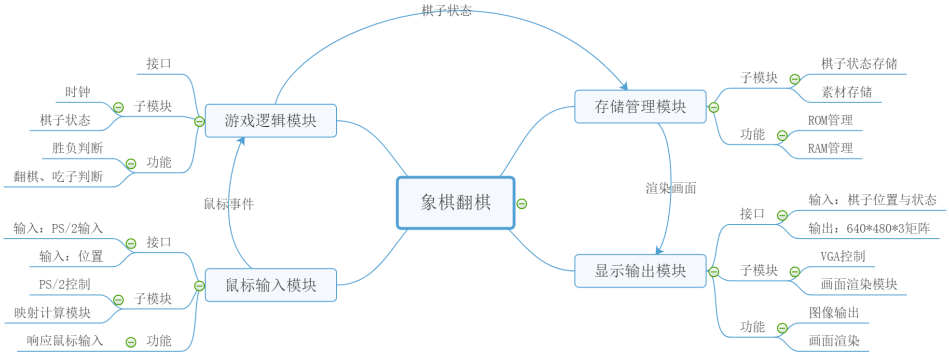
3. 存储管理模块

   负责读取 ROM，给游戏逻辑模块的数据管理提供底层支持，以及读取棋子和棋盘的贴图，给显示输出模块提供素材。

4. 显示输出模块

   包括一个底层接口和一个高层接口。底层接口负责利用 VGA 控制器将画面通过 VGA 接口输出，高层接口负责接受游戏逻辑模块提供的状态变化，同时结合存储管理模块提供的素材，给底层接口发出绘图的指令。

1.3 工作流程



1.4 任务分配

马昕宇：负责 IO 相关接口，以及管理存储设备等。

李晨昊：负责主体游戏逻辑的设计，以及设计游戏画面等。

2 关键技术分析

2.1 I/O 部分

1. ps2 鼠标输入

   鼠标输入部分比较简单，需要给逻辑部分传两个信号：点击与位置，这里的位置是指经过处理后对应棋盘上的位置，便于逻辑部分的处理。另外还需要给显示输出部分一个坐标信号，用来显示光标的位置。这部分主要都是在前人代码基础上修改而成的，因此难度并不大，但是也存在一定的问题，这在后面有详细的说明。

2. VGA 输出

   输出部分从逻辑部分收到棋盘盘面以及游戏状态、生命值等信息，并将其通过 VGA 显示到显示器上。只需要将棋盘的每一个位置遍历并将相应的棋子显示到该位置上即可，但是在棋盘较大时也出现过资源不足的情况。

3. 图片转 mif

   由于现有的转换工具效果较差，彩色图片的 RGB 各只有一位，因此我们自己使用 PIL 实现了一个将 jpg 图片转换为 mif 文件的工具。源码与使用方法已经通过邮件提交给老师。

2.2 逻辑部分

1. 翻棋和揭棋的逻辑

   总的来说逻辑复杂度揭棋 > 普通象棋 > 翻棋，从逻辑角度来讲实现起来都没有太大难度，只需要将类似于普通软件的代码放到合适的地方去即可。

   需要检测 reset 和鼠标点击来做出反应，我采取的策略是在时钟的上升沿检测，检测并处理后的一段时间内不再响应 reset 或鼠标点击，我选择的时间间隔是 0.3s，也就是说 0.3s 内连续点击两次会被当成一次，长按超过 0.3s 会被当成点击多次，这个时间是经过多次尝试得到的。正常的使用习惯下，基本不会有误判的情况。

   在响应信号时维护棋盘的状态，将状态传递给绘图模块即可显示出棋盘。连击和加倍等逻辑也在这个阶段处理：翻棋中吃对方的子可以增加倍数，连续吃三个以上对方的子，基础倍率会上升。

2. 随机棋盘生成

   我们使用了一个固定的初始随机数种子（其实很容易就可以改成用时间做种子），使用Xorshift方法来生成随机数，使用Fisher–Yates shuffle算法来生成随机序列。

虽然给随机数种子赋的初值可能是不可综合的，但是这对功能其实没有什么影响，上电之后是随机数也是可以接受的。

3 下载验证及演示说明

输入使用了 ps2 接口鼠标以及两个按键作为 reset，输出为 VGA 接口的显示器。两个按键分别为游戏重置与鼠标重置，按键位置如图所示。程序下载进去之后需要先 reset，之后重新开局也用 reset。

4 遇到的问题及解决方法

4.1 I/O 部分

1. FPGA 内存问题

   quartus2 定制 ROM 时对 mif 文件的 DEPTH 有限制，这导致图片的分辨率不能太高，此外 FPGA 的内存也只够保存很少的图片，因此在显示象棋盘时使用了一些技巧，利用棋盘的对称性，只保存棋盘的左上角，其余部分利用对称的方式显示。

2. 鼠标失灵问题

   鼠标部分用的是祖传代码，因此在使用时出现了一些小问题：鼠标偶尔会卡住不动。但这种情况出现的概率并不高，因此我们采用了单独对鼠标进行重置的方式来解决这个问题。

4.2 逻辑部分

1. 随机棋盘生成部分消耗资源过多

   现象是编译器使用内存超限，无法编译。值得注意的是，当我们在群里提出问题的时候，大家给我们参考的方向都是考虑 FPGA 的空间资源是否不足了，最终发现实际上这个问题应该是 FPGA 的逻辑资源不足了，中途走了一些弯路，这可以作为一个经验。

   解决方法是不在一个时钟周期内生成整个棋盘，而是在决定需要 reset 后，每过一个时钟周期执行一步。Fisher–Yates shuffle 算法中的一步很好定义，就是执行一次生成随机数及 swap。这样在数十个时钟周期后即可得到结果，这依然是很快的

2. 绘图部分消耗资源过多

   现象是编译卡死在 analysis & synthesis 阶段无法完成。

   经过多次尝试，发现如果只绘制一部分棋盘中的棋子，则资源要求可以满足，而揭棋棋盘中本来也不会全部占满棋子，只会有一小部分位置 (至多 32 个位置) 有棋子，所以可

以在逻辑部分维护一个当前有效的棋子数组，只绘制它们，而不是检查棋盘的每个位置看是否需要绘制。

此外，还曾经尝试过另一个方法：轮流绘制棋盘中的棋子。结果不理想：如果是在一次绘图中轮流绘制，则棋子上会出现条纹；如果是在多次绘图中轮流绘制，则棋子完全是模糊的。

5 实验总结与体会

通过这次实验，我们对 VHDL 的使用更加熟练了，对于 Quartus2 的使用方法也有了更深的了解。在调试的过程中虽然遇到了一些问题，但是经过反复的修改，所有的问题基本上都得到了解决。在这一过程中对于数字电路有了更深刻的理解，学会了如何在有限的资源下尽可能地将我们的设计完善，这对于之后计算机组成原理课程的学习应该会有一定的帮助。