



课程名称：数字逻辑设计

上课时间：秋冬周四 3,4,5;11,12

组长姓名：马恒瑞 3170106052

组员姓名：

罗炜程 3170105902,

孙浩 3170104834,

郭书廷 3170104871

指导教师: 王总辉, 洪奇军, 刘海风

数字逻辑设计期末课程设计实验报告

基于 FPGA 的金字塔纸牌消除游戏

2019 年 1 月 15 日

目录

| | |
|---------------------------|-----------|
| 1 项目简介与完成效果 | 2 |
| 1.1 游戏简介 | 2 |
| 1.2 操作方法 | 2 |
| 1.3 实现效果 | 3 |
| 2 游戏构架与模块设计 | 5 |
| 2.1 模块功能简介 | 5 |
| 2.1.1 输入部分 | 5 |
| 2.1.2 显示部分 | 5 |
| 2.1.3 图像处理部分 | 6 |
| 2.1.4 纸牌处理部分 | 6 |
| 2.2 模块关系结构图 | 6 |
| 3 代码实现与算法分析 | 8 |
| 3.1 实现详解 | 8 |
| 3.1.1 输入处理 | 8 |
| 3.1.2 纸牌处理 | 9 |
| 3.1.3 图像处理 | 10 |
| 3.1.4 VGA 显示 | 12 |
| 4 调试技巧与实验心得 | 14 |
| 4.1 调试中的问题 | 14 |
| 4.1.1 问题描述与原因分析 | 14 |
| 4.1.2 调试小结 | 15 |
| 4.2 技术亮点 | 16 |

| | | |
|----------|---------------------|-----------|
| 4.3 | 已知缺陷与改进空间 | 16 |
| 4.4 | 实验心得 | 17 |
| 4.4.1 | 设计的主要思路 | 17 |
| 4.4.2 | 编程与调试心得 | 17 |
| 4.4.3 | 总结 | 17 |
| 5 | 附录：核心代码 | 18 |
| 5.1 | Keyboard | 18 |
| 5.2 | keyActive | 19 |
| 5.3 | active | 21 |
| 5.4 | Bottom | 22 |
| 5.5 | Eliminate | 24 |
| 5.6 | FindPixel | 27 |
| 5.7 | vgac | 28 |

Chapter 1

项目简介与完成效果

1.1 游戏简介

本组设计游戏源于 windows10 自带的游戏《金字塔纸牌》。游戏规则如下：一副纸牌去除 Joker 共 52 张牌，其中 28 张牌按第一行一张，第二行两张...第七行七张的顺序依次排列成金字塔形（如图 1.1.0.1）。金字塔阵列中，第 $i+1$ 行压在第 i 行纸牌上方，即除第七行外，每张牌被两张牌压住，只有当一张牌上方没有其他纸牌时才能被消除。当选中的一张牌为 K 时，或者选中的两张牌数值之和为 13 时，可以被消除 (A, J, Q, K ，分别表示 1,11,12,13)。

金字塔阵列外的 24 张牌堆成一摞，每次选中中间按钮可以将左边的第一张牌移到右边（类似于两个栈）。左右两摞牌的第一张可以被选中进行消除。当左侧牌全部翻完之后，右侧的牌将全部回到左边。备用牌组至多可以翻三次，如果次数用完，游戏将失败。

1.2 操作方法

游戏使用外接键盘控制，用左右键控制光标，按空格键选中光标所在的纸牌。如果选中的最近的一张或两张纸牌面值和为 13，则消除纸牌，如果选中的是翻牌按钮，则进行一次翻牌，如果牌堆已空，则重新翻牌。



图 1.1.0.1: Windows10 上的金字塔纸牌

1.3 实现效果

如图 1.3.0.1、1.3.0.2是本组完成的金字塔游戏效果。

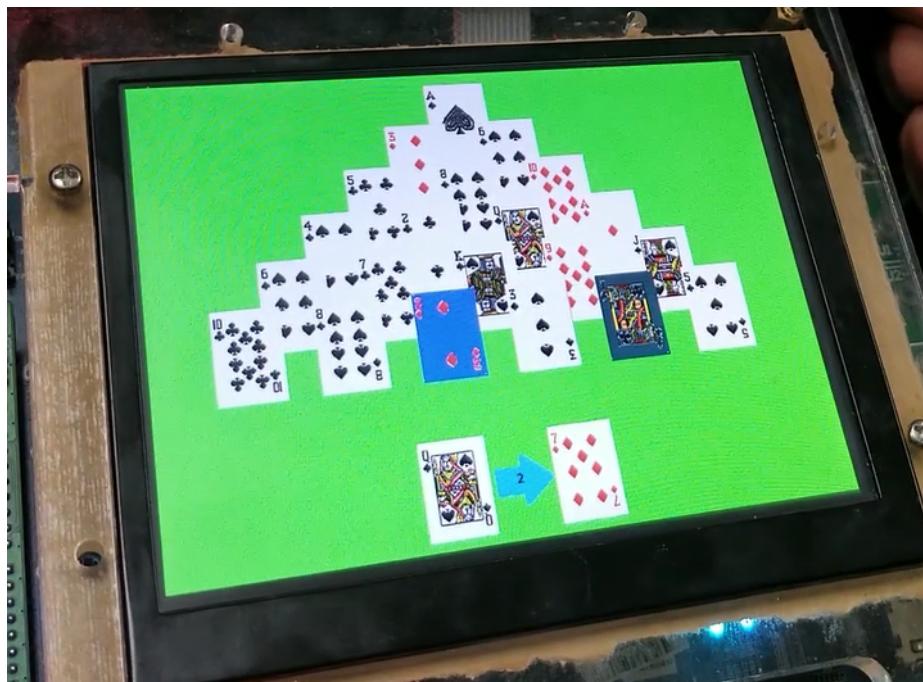


图 1.3.0.1: 本组实现的金字塔纸牌



图 1.3.0.2: 游戏通关与失败界面

Chapter 2

游戏构架与模块设计

2.1 模块功能简介

2.1.1 输入部分

Keyboard 模块，直接与键盘硬件输入的信号相连，将硬件信号转化成 KeyActive 模块可以理解的信息。

KeyActive 执行键盘信号，处理光标移动和选中指令，将选中的牌传输给纸牌处理模块。

2.1.2 显示部分

直接与显示硬件相连的模块有两个，vgac 和 DispNum。vgac 模块负责 VGA 显示屏主界面输出，DispNum 负责七段数码管显示，用于显示左右两个牌堆的纸牌数量。

VGAC 屏幕采用 4 位 rgb 显示，像素刷新频率为 25MHz，模块会逐行扫描像素点，显示有效范围内的像素。

DispNum 使用课程作业中常用的 MC14495 模块。

2.1.3 图像处理部分

FindPixel 设计使用一个 FindPixel 模块，用于在线查询对于一个给定屏幕坐标 (X, Y) 的像素属于哪张牌，并给出这一像素点对于这张牌的相对坐标 (dX, dY) 。

该模块会调用 **FindBlock** 模块，这一模块会检查各张牌是否被消除，检查纸牌的遮挡关系，从而确定像素属于哪张纸牌。

2.1.4 纸牌处理部分

Active 检查一张纸牌是否可以被选取，即该纸牌的上方有没有被其他纸牌覆盖。

Eliminate 消除模块，如果选中的排可以消除，则更新纸牌序列。

Bottom 用于处理显示在屏幕底部的牌堆，执行翻牌操作。

2.2 模块关系结构图

如图 2.2.0.1 展示了本组游戏设计的基本构架图（包含各个模块间的信息交流）。

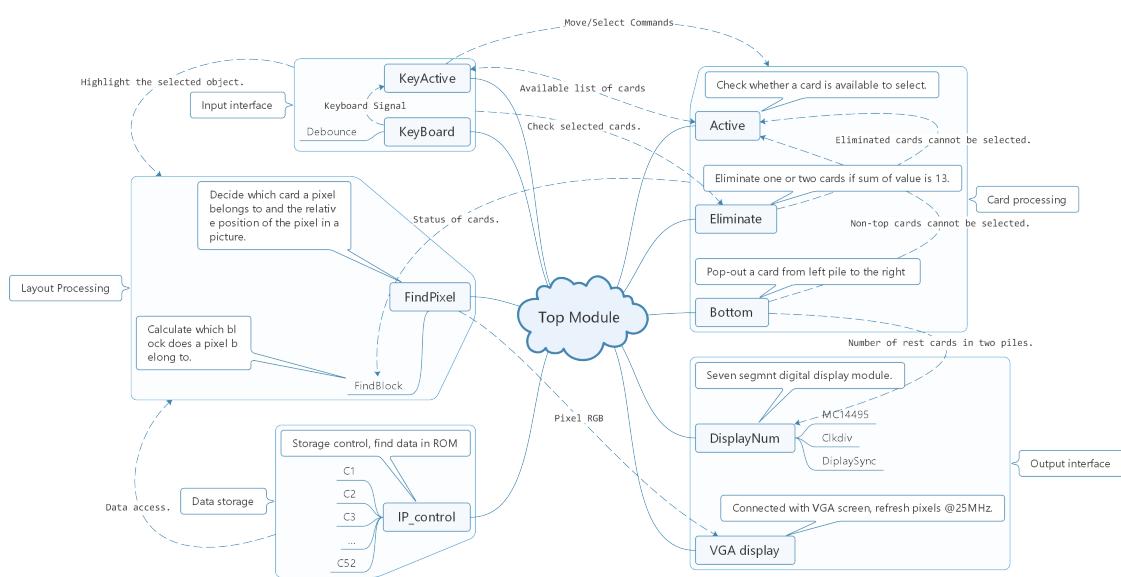


图 2.2.0.1: 模块构架图 (虚线表示模块间通过 top 传递的信息)

Chapter 3

代码实现与算法分析

3.1 实现详解

3.1.1 输入处理

输入使用 PS2 协议（美式英文布局）键盘，PS2 键盘传输协议见表 3.1.1.1。

| 位数 | 功能 | 数据 |
|----|-----|--------------|
| 1 | 起始位 | 0 |
| 8 | 数据位 | (LSB) 低位在前 |
| 1 | 校验位 | 奇校验 |
| 1 | 停止位 | 1 |
| 1 | 应答位 | 仅用于对主机设备的通讯中 |

表 3.1.1.1: PS2 键盘传输协议

在键盘通信时 8 位数据位是描述键盘按键的关键数据位。每个按键都分为按下和抬起两次发送信号，每个按键有一个扫描码，所有按键扫描码构成一套扫描码集，目前一共有三套扫描码集，键盘总通常使用第二套扫描码集。

键盘上有些按键需要额外一个字节的扩展码，例如方向左键的扫描码为 *E06B*，也就是说这些按键的扫描码将分两次传输。当按键按下时键盘发送

通码，按键抬起时发送断码。通常断码是在通码中再加入一个字节的 *F0*。

本次设计中共有三个有效按键分别是方向左键，方向右键和空格键。其扫描通码为 *E0 6B, E0 72, 29*，断码为 *E0 F0 6B, E0 F0 72, E0 29*。另外设计对键盘信号输入提供了防误触发的 debounce 处理，每个按键需要按下足够长的时间才能被触发。

键盘处理为 **Keyboard** 模块，代码见附录 5.1.0.1。该模块输出三个按键的状态，并提交给 **keyActive** 模块。当左右键被按下时，更新当前光标移动方向，并按照移动方向依次从上到下，从左至右依次在有限状态机中检测可选取的纸牌，直到找到可选取的纸牌，将光标设置在该纸牌上，并高亮显示。当按下空格键时，选取当前纸牌，如果当前纸牌不可用，则找到下一个可用纸牌。

keyActive 模块代码见附录 5.2.0.1。

3.1.2 纸牌处理

界面上共有 30 张纸牌和一个翻牌按钮。共有 31 个可选取的块，分别编号 $1, 2, \dots, 31$ ，如图 3.1.2.1。首先，一张可选取的牌应满足该张牌上方没有其他牌，有如下几种情况

1. 一张牌位于第七行且未被消除。
2. 一张牌位于前六行未被消除，且覆盖于其上的两张牌均被消除。
3. 位于 29 或 31 号位置上，且未被消除。
4. 位于 30 号位置上。

对于第二种情况，只需要枚举每一张牌即可，如 13 号位置的牌应满足

$$State_{13} = \sim State_{18} \& \sim State_{19}$$

对于 1-28 号纸牌，使用 **active** 模块来判断每张纸牌是否可以被选取。5.3.0.1 是 active 模块代码。对于 29 和 31 号纸牌，我们认为他们始终可选。而对于 30 号的翻牌器，则使用 **Bottom** 模块来处理。

Bottom 模块首先记录一个预设的牌组序列，并使用寄存器记录状态。*left* 表示当前左侧（29 号）的第一张牌，*right* 表示当前右侧（31 号）的第一张牌，*next-left* 表示左侧将要显示的牌，*next-right* 表示右侧将要显示的牌。每当 **keyActive** 选中 30 号翻牌器时，**Bottom** 模块自动读取下一张牌，用

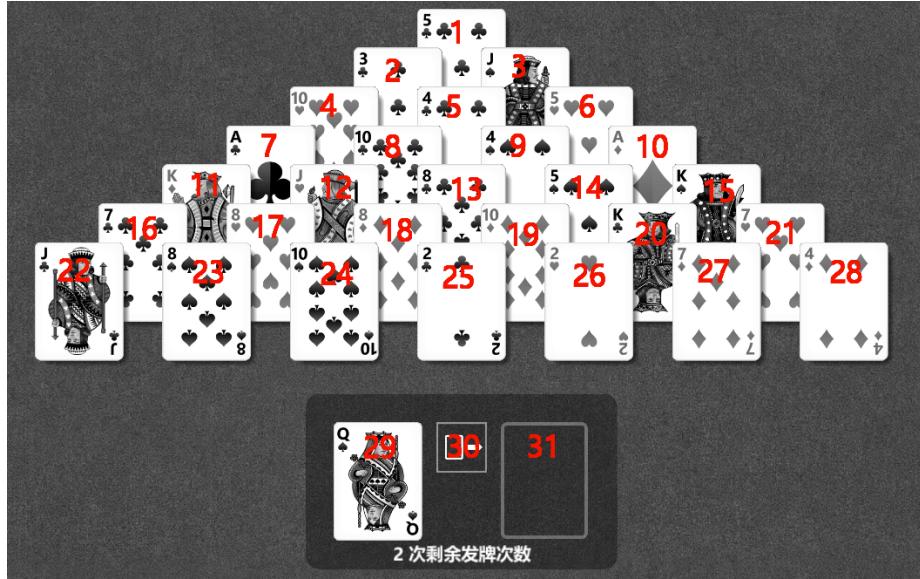


图 3.1.2.1: 可选取块的编号

`f,next-f` 两个寄存器记录位置，同时更新 `next-left,next-right`。同时 `Bottom` 模块按时钟每 $1.3\text{ms}(1/2^{16}\text{s})$ 将 `next` 寄存器的值更新，同时显示模块也会因寄存器值的改变而刷新显示。

`Bottom` 模块代码见附录 5.4.0.1。

如果 `keyActive` 模块选中的不是 30 号，则说明玩家选中了一张纸牌，同时，选中纸牌意味着可能发生消除。所以当空格键按下，`keyActive` 模块输出的 `spacedown` 的上升沿将同时触发 **Bottom** 和 **Eliminate** 模块。

`Eliminate` 模块将检查最近选择的两张纸牌，如果最近选择的一张是 K，或者最近选择的两张值加起来等于 13，则执行消除操作：1. 标记该牌为已消除状态；2. 标记该牌为不可选择状态；3. 更新下一个光标的位置，并在下一个时钟周期内刷新显示光标位置。

`Bottom` 模块代码见附录 5.5.0.1。

3.1.3 图像处理

图像处理主要是 `FindPixel` 模块，该模块会返回屏幕坐标为 (X,Y) 的点属于哪张牌，并返回该像素点在这张牌上的相对位置 (dX,dY) 。随后，在 `top` 模块中调用 `ipControl` 模块从而读取图片相应像素的 RGB 值。(图片需

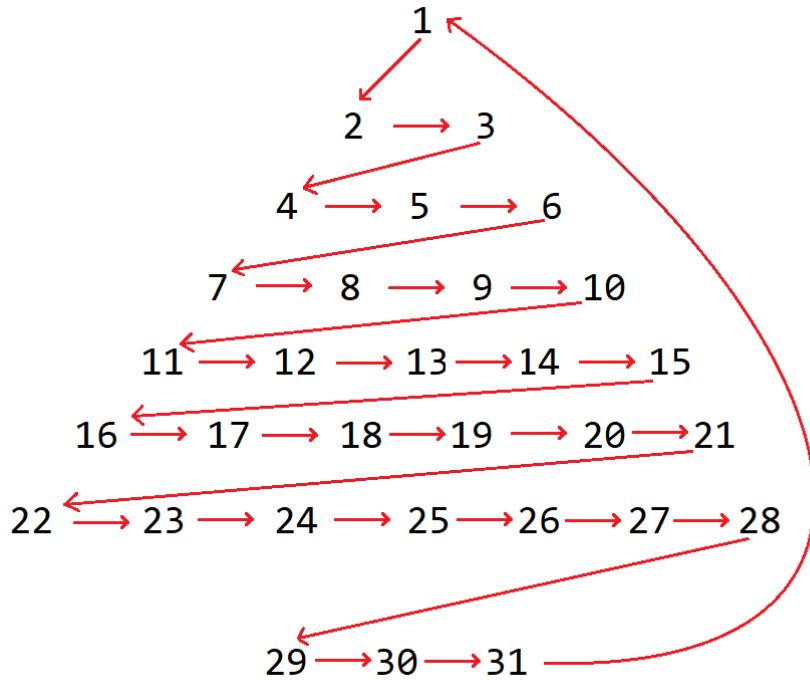


图 3.1.2.2: 状态转移顺序

要预先使用转换器及脚本将 png 文件转换成 coe 文件)

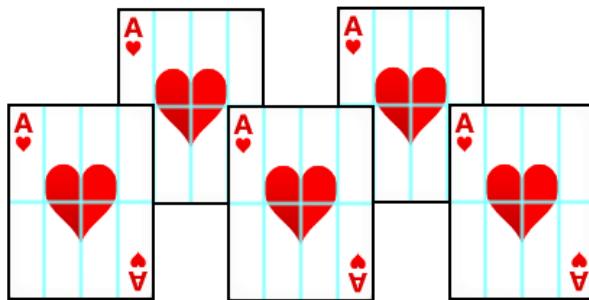


图 3.1.3.1: 将每张纸牌分为 8 块

为了方便计算和显示美观，我们将一张牌分为纵向 4 块，横向两块，共 8 块，如图 3.1.3.1，每张牌左右被覆盖四分之一，上下被覆盖二分之一。这样可以将整个版面分为 $40 * 8 = 320$ 块，每块高 44px，宽 15px。按如下代

码即可计算出坐在块的编号。

Listing 3.1.3.1 FindBlock

```
1: row = (Y - 20)/44;  
2: column = (X - 20)/15;  
3: BlockNum = column + row*40;
```

Findpixel 模块将绝对坐标转化为一张牌内的相对坐标，代码见附录 5.6.0.1。

3.1.4 VGA 显示

VGA 显示的信号输入主要有以下几个：时钟、RGB、水平位置、垂直位置。将这些信号传给如下模块，模块将按时钟输出同步水平扫描、垂直扫描和 RGB。只要按照一定频率每次传入像素位置和颜色，该模块就可以通过水平和垂直同步信号将指定的像素点刷新到显示屏上。其原理如下。

由于显示屏不断接受横向和纵向的扫描信号，并按照信号改变刷新像素的位置。当扫描位置刚好处于要刷新的位置时，设置 rdn 输出让显示屏读取颜色 RAM，并将颜色 RAM 输出为指定的 RGB，显示屏将改变当前像素的内容。否则将 rdn 输出为不刷新当前像素（在此模块中 0 表示刷新，1 表示不刷新），像素将保留原有颜色。如图 3.1.4.1

本设计种使用的是 $525*800=420k$ 的显示屏，像素扫描频率为 25MHz，屏幕刷新频率为 59.5Hz，即屏幕秒刷新约 60 次，由人眼的视觉暂留效应，就可以看到连续的动画。

VGA 显示模块代码见附录 5.7.0.1。

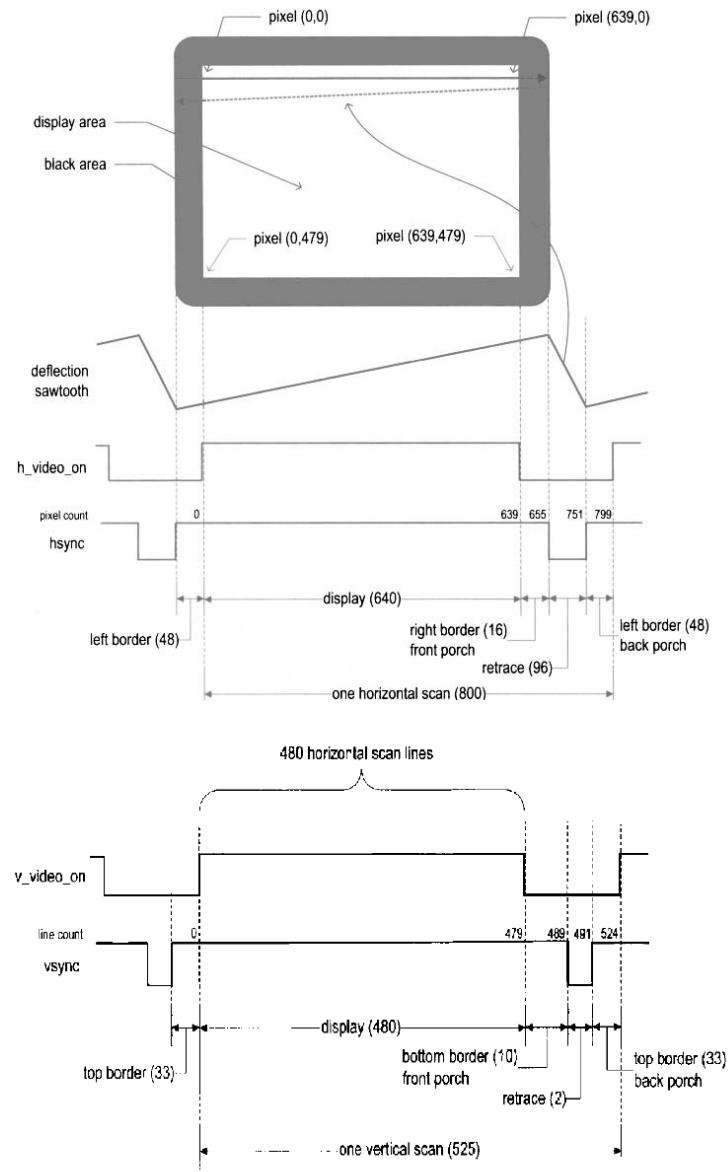


图 3.1.4.1: VGA 显示原理

Chapter 4

调试技巧与实验心得

4.1 调试中的问题

4.1.1 问题描述与原因分析

调试中遇到了各式各类的问题，此处仅挑选其中相对比较有代表性的几个例子。

VGA 显示异常

问题描述 VGA 显示屏无法按照预期显示正常的图像，出现错位、不显示等问题。

问题原因 VGA 模块的像素 RGB 高低位顺序写反、同步扫描信号周期频率与屏幕的分辨率不符

纸牌错位

问题描述 纸牌显示错位，或被拆分成若干块。

问题原因 像素坐标计算错误，ip 核访问方法出错。

纸牌层叠关系错误

问题描述 纸牌层叠的顺序错误，个别应该被盖住的纸牌出现在顶层等。

问题原因 记录纸牌当前状态的寄存器组存在时序错误，具体原因是混用 \Leftarrow 和 $=$ ，导致赋值语句混乱。

键盘操作异常

问题描述 操作过程中键盘按键出现误触发，或移动方向相反的问题。

问题原因 键盘模块处理错误、没有加入防抖动模块。

消牌异常

问题描述 消牌异常，选取的纸牌在不该消除时消除，在该消除时不消除。

问题原因 同纸牌层叠关系错误的原因，同时还有高亮纸牌与实际选取的纸牌不符的问题。

选牌错误

问题描述 误选中不可以被选择的牌，选牌时移动的顺序错误。

问题原因 在按下 space 键选牌的时候，有可能光标正在牌组间转移，如果直接选定当前牌则会跳过可选择性检查。

引脚不足的问题

问题描述 生成 bit 文件时出现引脚不足的问题。

问题原因 使用的 ip 核过多，存储的信息过多，导致板载空间不足。最终通过优化存储结构、减少变量等方法，解决了该问题。

4.1.2 调试小结

1. 调试中出现频率最高的是时序问题，可见时序电路在设计和实现上相较于组合逻辑电路都有更高的复杂性和难度。
2. 总体来说，调试过程相比 c 程序设计这种可以单步调试的方法，通过输出、反复手工验证的方法调试非常耗费时间。另外，尽管我组使用

了最新版的 vivado18.3，ISE 设计软件本身对工程编译一次的时间仍然很长，导致课程项目的完成日期不断延后。

3. 通过调试代码，我们发现，尽管很多我们认为理所当然的操作，也可能需要较为复杂的软件支持才能达到预期目的。（比如键盘加入防抖动模块）

4.2 技术亮点

1. 输入方面，本组使用 PS2 协议的键盘作为输入方式，相比板载按键具有更好的易用性，且在处理信号上需要代码支持，提升了设计的难度。
2. 输出方面，本组设计的游戏中纸牌存在层叠关系，相比其他飞机大战、推箱子等应用，需要更为复杂的显示算法。另外，对纸牌高亮显示需要对图形处理进行精心设计。
3. 存储方面，本组设计中使用了大量的 ip 核，使用一个独立的模块用于管理 ROM 访问，信息存储的管理是一大亮点。
4. 游戏设计方面，游戏规则相对较为复杂，界面的有效按键数众多，设计状态机时需要 4 个以上的状态机以满足游戏规则的需要。
5. 项目构架方面，本组在模块功能的设计上充分考虑各模块复杂度的均衡程度，在工程构架上具有清晰明确的结构，即控制了代码长度，有简化了调试时间（见 2.2 图 2.2.0.1）。

4.3 已知缺陷与改进空间

1. 键盘防抖动模块效果并不完善，有时需要按下较长时间才能触发。
2. 长按按键不能自动移动光标，必须一下一下按。
3. 牌组只能预设，不能随机生成。（原计划写一个线性同余法的伪随机数生成器用于生成纸牌，后来考虑时间不足，以及随机寻址等功能实现的复杂性，放弃了该计划。）
4. 纸牌图片质量较差，看起来视觉效果不佳，另外，界面背景可以添加图片以代替纯色背景。

4.4 实验心得

4.4.1 设计的主要思路

一开始的想法是，这个游戏比较静态，实现起来难度应该不大。每次消除时对变化部分进行更新即可（局部操作）。但后来发现，Verilog 和 VGA 的逻辑和预想不太一样。实际上大部分的参数和操作都是直接与对应模块连接的，需要维护每个分量的状态。

不过最终设计还是基本保持了最初的想法，在状态机设计中主要都是保持操作。尽管逻辑不难，但是遇到了很多匪夷所思的时序问题，导致工期很长。

4.4.2 编程与调试心得

总的来说，我们由于对 Verilog 的了解尚浅，运用不够熟练，在设计时为了避免复杂操作导致的冲突问题，我们使用了大量的模块与 IP 核。为了方便调试，我们使每个模块的代码量都在 100 行以内，只求实现一个基本的功能，之后再进行组装。这样假如出现了 BUG 也方便及时修复，降低了成本。虽然还是有模块单独工作正常，但组装后却出错的情况，但调试这类问题相比调试一个模块里的复杂问题，解决各模块的协作问题显得相对简单。

此外，利用 IP 核储存一些频繁使用的信息（如图片），减少在 Verilog 程序中的计算，判断等过程。然而缺点是在生成执行文件时比较耗时间，而且在后期加入封面等图片时出现了储存 IP 核用的引脚不够的问题。

4.4.3 总结

1. 在项目初期，目标不要订得太高，先实现一个小功能。否则，一个复杂模块容易出现很多难以排查的错误。
2. 在组合不同模块时，要注意统一不同模块之间的时序，不要彼此冲突。在逻辑无误的情况下出现错误，首先应该想到时序问题。
3. 对于一个功能的实现没必要在一棵树上吊死，多换几种写法，说不定就对了。
4. 大作业一定要早点开始，完成一个设计不仅需要脑力，还需要充足的时间和体力支持。

Chapter 5

附录：核心代码

5.1 Keyboard

Listing 5.1.0.1 Keyboard

```
1: module Keyboard(
2:   input wire clk_25MHz,
3:   input wire PS2Clk,
4:   input wire PS2Data,
5:   output reg upKeyState,
6:   output reg downKeyState,
7:   output reg leftKeyState,
8:   output reg rightKeyState,
9:   output reg spaceKeyState);
10:
11: wire debouncePS2Clk;
12: Debounce m_Debounce(.debounceClk(clk_25MHz),
13: .button(PS2Clk), .debounceButton(debouncePS2Clk));
14:
15: initial begin upKeyState = 1'b0;
16: downKeyState = 1'b0;
17: leftKeyState = 1'b0;
18: rightKeyState = 1'b0;
19: spaceKeyState = 1'b0; end
20:
```

```

21:   reg [7:0] key;
22:   reg extendFlag, endFlag;
23:   initial begin extendFlag = 1'b0;
24:   endFlag = 1'b0; end
25:
26:   reg [3:0] cnt;
27:   initial cnt = 4'd0;
28:   //nege to show that we have already pressStateed
29:   always @(posedge debouncePS2Clk) begin
30:     if (cnt >= 4'd1 && cnt <= 4'd8)
31:       key[cnt - 1] = PS2Data;
32:     cnt = cnt + 4'd1;
33:     if (cnt == 4'd11) begin
34:       cnt = 4'd0;
35:       //extend to indicate the space
36:       if (key == 8'hE0) extendFlag = 1'b1;
37:       else if (key == 8'hF0) endFlag = 1'b1;
38:       else begin
39:         //assigning state
40:         //negative logic
41:         if (key == 8'h75) upKeyState = ~endFlag;
42:         else if (key == 8'h72) downKeyState = ~endFlag;
43:         else if (key == 8'h6B) leftKeyState = ~endFlag;
44:         else if (key == 8'h74) rightKeyState = ~endFlag;
45:         else if (key == 8'h29 && extendFlag == 1'b0)
46:           spaceKeyState = ~endFlag;
47:         extendFlag = 1'b0;
48:         endFlag = 1'b0;
49:       end end end
50:   endmodule

```

5.2 keyActive

Listing 5.2.0.1 keyActive

```

1: module keyActive(
2:   input wire key_left, key_right,
3:   input wire [31:0] clkdiv,

```

```

4:   input wire[31:0] enable,
5:   output reg [5:0] idx
6: );
7:
8: localparam LEFT = 2'b10, RIGHT = 2'b01, STILL = 2'b00;
9: reg [1:0] state, state_next;
10: reg [5:0] idx_next;
11: reg last23;
12: initial begin state=STILL; end
13: //Notice that a order is availabel only when a
14: // key is pressed long enough
15: always@(posedge clkdiv[15]) begin
16:   if (last23==0 && clkdiv[23]==1) begin
17:     idx <= idx_next;
18:     state <= state_next;
19:   end
20:   last23<=clkdiv[23];
21:   if (~enable[idx]) begin
22:     idx <= idx_next;
23:     state <= state_next;
24:   end end
25: //Then assign corresponding value for orders
26: // according to keyboard code
27: always@(*) begin
28:   case (state)
29:     LEFT: begin
30:       if(key_right) begin
31:         state_next <= RIGHT;
32:         idx_next <= idx;
33:       end
34:       else begin
35:         state_next <= LEFT;
36:         if(idx == 6'd31)
37:           idx_next <= 6'b1;
38:         else
39:           idx_next <= (~enable[idx] | key_left) ?
40:             idx + 1 : idx;
41:       end
42:     end

```

```

43:     RIGHT: begin
44:         if(key_left) begin
45:             state_next <= LEFT;
46:             idx_next <= idx;
47:         end
48:         else begin
49:             state_next <= RIGHT;
50:             if(idx == 6'b1)
51:                 idx_next <= 6'd31;
52:             else
53:                 idx_next <= (~enable[idx] | key_right) ?
54:                     idx - 1 : idx;
55:             end
56:         end
57:     STILL: begin
58:         if (key_left) begin
59:             state_next <= LEFT;
60:             idx_next <= idx;
61:         end
62:         else if (key_right) begin
63:             state_next <= RIGHT;
64:             idx_next <= idx;
65:         end
66:         else begin
67:             state_next <= STILL;
68:             idx_next <= idx;
69:         end
70:     end
71:     endcase
72: end
73: endmodule

```

5.3 active

Listing 5.3.0.1 active

```

1: module active(
2:     input wire [52:0] state,

```

```

3:   output wire [28:0] temp
4: );
5: assign temp[1]=(~state[2])&(~state[3])&state[1];
6: assign temp[2]=(~state[4])&(~state[5])&state[2];
7: assign temp[3]=(~state[6])&(~state[5])&state[3];
8: assign temp[4]=(~state[7])&(~state[8])&state[4];
9: assign temp[5]=(~state[8])&(~state[9])&state[5];
10: assign temp[6]=(~state[9])&(~state[10])&state[6];
11: assign temp[7]=(~state[11])&(~state[12])&state[7];
12: assign temp[8]=(~state[12])&(~state[13])&state[8];
13: assign temp[9]=(~state[13])&(~state[14])&state[9];
14: assign temp[10]=(~state[14])&(~state[15])&state[10];
15: assign temp[11]=(~state[16])&(~state[17])&state[11];
16: assign temp[12]=(~state[17])&(~state[18])&state[12];
17: assign temp[13]=(~state[18])&(~state[19])&state[13];
18: assign temp[14]=(~state[19])&(~state[20])&state[14];
19: assign temp[15]=(~state[20])&(~state[21])&state[15];
20: assign temp[16]=(~state[22])&(~state[23])&state[16];
21: assign temp[17]=(~state[23])&(~state[24])&state[17];
22: assign temp[18]=(~state[24])&(~state[25])&state[18];
23: assign temp[19]=(~state[25])&(~state[26])&state[19];
24: assign temp[20]=(~state[26])&(~state[27])&state[20];
25: assign temp[21]=(~state[27])&(~state[28])&state[21];
26: assign temp[22]=state[22];
27: assign temp[23]=state[23];
28: assign temp[24]=state[24];
29: assign temp[25]=state[25];
30: assign temp[26]=state[26];
31: assign temp[27]=state[27];
32: assign temp[28]=state[28];
33: endmodule

```

5.4 Bottom

Listing 5.4.0.1 Bottom

```

1: module Bottom(
2:   input wire[31:0] clkdiv,

```

```

3:   input wire[5:0] select,
4:   input wire spacedown,
5:   input wire [25:0] enable,
6:   output reg[5:0] left,
7:   output reg[5:0] right,
8:   output reg[5:0] f,
9:   output reg fail
10: );
11: initial begin left=6'd24; right=6'd25; f=55; end
12: reg [5:0] next_left,next_right,next_f;
13: always @(*) begin
14:   if(spacedown && select==6'd30) begin
15:     if(left==6'b1) begin
16:       next_right=6'd25;
17:       next_left=6'd24;
18:       next_f=f-6'd1;
19:     end
20:     else begin
21:       next_right=left;
22:       next_left=left-6'd1;
23:       next_f=f;
24:     end
25:   end
26:   else begin
27:     next_left=(~enable[left] && left!=6'b0)?
28:               left-6'd1:left;
29:     next_right=(~enable[right] && right!=6'd25)?
30:                 right+6'd1:right;
31:     next_f=f;
32:   end
33: end
34: reg last23;
35: always @(posedge clkdiv[15]) begin
36:   if (~enable[left] || (~enable[right] && right!=6'd25))
37:   begin
38:     left <= next_left;
39:     right<= next_right;
40:     f<=next_f;
41:   end

```

```

42:     else if (last23==0 && clkdiv[23]==1) begin
43:         left <= next_left;
44:         right<= next_right;
45:         f<=next_f;
46:     end
47:     last23=clkdiv[23];
48: end
49: endmodule

```

5.5 Eliminate

Listing 5.5.0.1 Eliminate

```

1: module Eliminate(
2:     input wire[31:0] clkdiv,
3:     input wire[4:0] value,
4:     input wire spacedown,
5:     input wire[4:0] position,
6:     input wire[31:0] enable,
7:     input wire[5:0] Left,
8:     input wire[5:0] Right,
9:     output reg[52:0] Cards,
10:    output reg[5:0] CurrentScore,
11:    output reg[4:0] CurrentPosition,
12:    output reg[4:0] CurrentValue,
13:    output reg[4:0] CurrentState
14: );
15: //several states are defined, to establish an FSM
16: reg[52:0] NextCards;
17: reg[5:0] NextScore;
18: reg[4:0] NextValue;
19: reg[5:0] NextPosition;
20: reg[4:0] nextState;
21: initial begin
22:     Cards = ~ (53'b0);
23:     NextCards = ~ (53'b0);
24:     currentState = 1'b0;
25:     currentValue = 4'b0;

```

```

26:   CurrentPosition = 4'b0;
27:   CurrentScore = 6'b0;
28:   NextScore = 6'b0;
29:   end
30:   //Next states calculation. Genarally,
31:   // if the selected card(s) have a sum 13,
32:   // they will be eliminated
33:   always @(*) begin
34:     if (spacedown && (enable[position]) &&
35:         position!=30 && (~ (position==31 && Right==25)))
36:       begin
37:         if (CurrentState==5'b1 && position!=CurrentPosition)
38:           begin
39:             if (CurrentValue + value == 5'd13) begin
40:               NextPosition = position;
41:               NextCards = Cards;
42:               if (position==29)
43:                 NextCards[Left]=1'b0;
44:               else if (position==31)
45:                 NextCards[Right]=1'b0;
46:               else
47:                 NextCards[position]=1'b0;
48:
49:               if (CurrentPosition==29)
50:                 NextCards[Left]=1'b0;
51:               else if (CurrentPosition==31)
52:                 NextCards[Right]=1'b0;
53:               else
54:                 NextCards[CurrentPosition]=1'b0;
55:
56:               NextScore = CurrentScore + 6'd2;
57:               nextState = 5'b0;
58:               NextValue = value;
59:             end
60:           else begin
61:             NextPosition = position;
62:             NextCards = Cards;
63:             NextScore = CurrentScore;
64:             nextState = 5'b0;

```

```

65:           NextValue = value;
66:       end
67:   end
68: //renew the current states
69: else begin
70:     if (value == 5'd13) begin
71:         NextPosition = position;
72:         NextCards = Cards;
73:         if (position==29)
74:             NextCards[Left]=1'b0;
75:         else if (position==31)
76:             NextCards[Right]=1'b0;
77:         else
78:             NextCards[position]=1'b0;
79:
80:         NextScore = CurrentScore + 6'd1;
81:         NextState = 5'b0;
82:         NextValue = value;
83:     end
84:     else begin
85:         NextPosition = position;
86:         NextCards = Cards;
87:         NextScore = CurrentScore;
88:         NextState = 5'b1;
89:         NextValue = value;
90:     end
91:   end
92: end
93: else begin
94:     nextState <= currentState;
95:     NextPosition <= currentPosition;
96:     NextValue <= currentValue;
97:     NextCards = Cards;
98:     NextScore <= CurrentScore;
99:   end
100: end
101: reg last23;
102: always @(posedge clkdiv[15]) begin
103:   if (!spacedown) begin

```

```

104:     CurrentState <= NextState;
105:     currentPosition <= NextPosition;
106:     CurrentValue <= NextValue;
107:     CurrentScore <= NextScore;
108:     Cards <= NextCards;
109:   end
110: else if (clkdiv[23]==1 && last23==0) begin
111:     CurrentState <= NextState;
112:     currentPosition <= NextPosition;
113:     CurrentValue <= NextValue;
114:     CurrentScore <= NextScore;
115:     Cards <= NextCards;
116:   end
117: last23=clkdiv[23];
118: end
119: endmodule

```

5.6 FindPixel

Listing 5.6.0.1 FindPixel

```

1: module FindPixel(
2:   input wire[9:0] X,
3:   //given its global position
4:   input wire[8:0] Y,
5:   //given the state of all cards(present or not)
6:   input wire[52:0] Exist,
7:   input wire clk,
8:   //provide a serie number from 0-32
9:   output wire[4:0] CardNum,
10:  output wire[6:0] dX,
11:  //relative position
12:  output wire[6:0] dY
13: );
14: //the firstly expected card
15: wire[7:0] First;
16: //the secondly expected card(for overlap exists)
17: wire[7:0] Second;

```

```

18: //the number of selected block
19: wire [8:0] BlockNum;
20: wire [9:0] rX;
21: wire [8:0] rY;
22: //r is the position of selected card's first pixel
23: wire [23:0] r;
24:
25: //Find selected block
26: FindBlock m0(X,Y,clk,BlockNum);
27: first m1(BlockNum,First);
28: second m2(BlockNum,Second);
29: assign CardNum = (X>=20&X<=619&Y>=20&Y<=371)?
30: ((Exist[First]==1'b1)?
31: First[4:0]:((Exist[Second]==1'b1)?
32: Second[4:0]:0)):((Y>=383&Y<=470)?
33: ((X>=230&X<=289)?
34: 5'd29:((X>=290&X<=349)?
35: 5'd30:((X>=350&X<=409)?5'd31:0))):0);
36: //refer to the IP core to find r
37: Reference r0({1'b0,CardNum},r);
38: assign rX = r[21:12];
39: assign rY = r[8:0];
40: assign dX = X - rX;
41: //relative position
42: assign dY = Y - rY;
43: endmodule

```

5.7 vgac

Listing 5.7.0.1 vgac

```

1: module vgac
2: (vga_clk,clrn,d_in,row_addr,col_addr,rdn,r,g,b,hs,vs);
3: // bbbb_gggg_rrrr, pixel
4: input [11:0] d_in;
5: // 25MHz
6: input vga_clk;
7: input clr_n;

```

```

8: // pixel ram row address, 480 (512) lines
9: output reg [8:0] row_addr;
10: // pixel ram col address, 640 (1024) pixels
11: output reg [9:0] col_addr;
12: // red, green, blue colors
13: output reg [3:0] r,g,b;
14: // read pixel RAM (active_low)
15: output reg      rdn;
16: // horizontal and vertical synchronization
17: output reg      hs,vs;
18:
19: // h_count: VGA horizontal counter (0-799)
20: // VGA horizontal counter (0-799): pixels
21: reg [9:0] h_count;
22: always @ (posedge vga_clk) begin
23:     if (!clrn) begin
24:         h_count <= 10'h0;
25:     end else if (h_count == 10'd799) begin
26:         h_count <= 10'h0;
27:     end else begin
28:         h_count <= h_count + 10'h1;
29:     end
30: end
31: // v_count: VGA vertical counter (0-524)
32:
33: // VGA vertical counter (0-524): lines
34: reg [9:0] v_count;
35: always @ (posedge vga_clk or negedge clrn) begin
36:     if (!clrn) begin
37:         v_count <= 10'h0;
38:     end else if (h_count == 10'd799) begin
39:         if (v_count == 10'd524) begin
40:             v_count <= 10'h0;
41:         end else begin
42:             v_count <= v_count + 10'h1;
43:         end
44:     end
45: end
46: // signals, will be latched for outputs

```

```

47:     wire [9:0] row=v_count - 10'd35; // pixel ram row addr
48:     wire [9:0] col=h_count - 10'd143; // pixel ram col addr
49:     wire h_sync = (h_count > 10'd95); // 96 -> 799
50:     wire v_sync = (v_count > 10'd1); // 2 -> 524
51:     wire read = (h_count > 10'd142) && // 143 -> 782
52:                 (h_count < 10'd783) && // 640 pixels
53:                 (v_count > 10'd34) && // 35 -> 514
54:                 (v_count < 10'd515); // 480 lines
55:     // vga signals
56:     always @ (posedge vga_clk) begin
57:         row_addr <= row[8:0]; // pixel ram row address
58:         col_addr <= col; // pixel ram col address
59:         rdn <= ~read; // read pixel (active low)
60:         hs <= h_sync; // horizontal synchronization
61:         vs <= v_sync; // vertical synchronization
62:         r <= rdn ? 4'h0 : d_in[3:0]; // 3-bit red
63:         g <= rdn ? 4'h0 : d_in[7:4]; // 3-bit green
64:         b <= rdn ? 4'h0 : d_in[11:8]; // 2-bit blue
65:     end
66: endmodule

```
