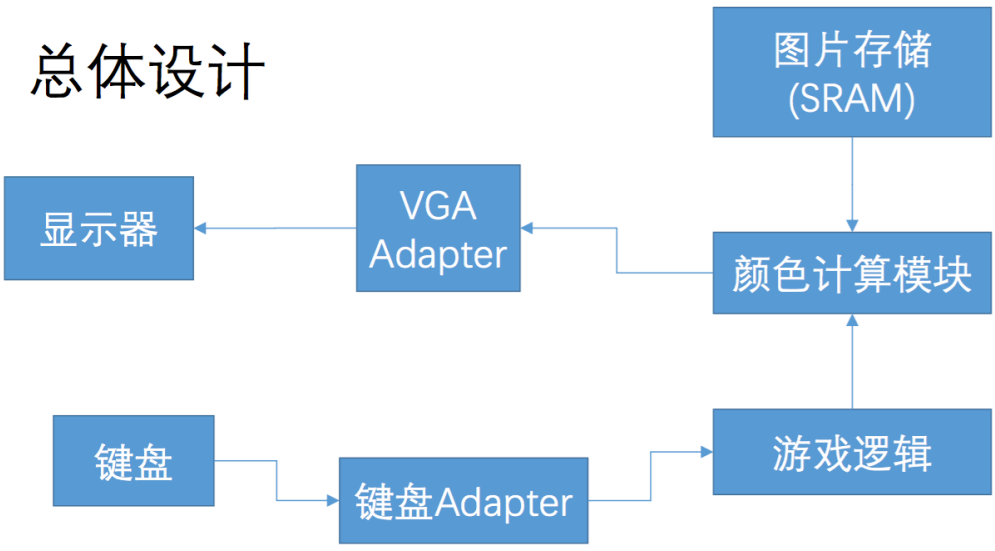
基于 FPGA 的拳皇游戏

2014011333 丁铭
2014010257 杜家驹
2014011342 罗干

一、项目介绍

本项目参考了经典格斗游戏拳皇，在 FPGA 中实现了拳皇中的八神庵和草稚京两个角色，在游戏性上也实现了拳皇的简化版的操作。
在项目根目录下包括以下一些文件和文件夹：
buildtxt 文件夹下是一个 visual studio2012 的 c++代码工程，需要在 win7 64 位系统下运行，负责处理素材生成二进制文件 data，由于需要读取 bmp 文件，所以需要使用 opencv 库，版本 2.4.10，但 opencv 太大没有放进来；另外在该工程中有 27 张图片素材，通过该工程转成二进制文件写入 sram。
project 文件夹下是拳皇的 FPGA 工程代码，其中的 output_files 下有编译好的 sof 和 pof 文件。
data 和 pre.pdf 分别是写入 sram 的二进制文件和展示文件，还有一个就是本文档是报告文件。

二、总体设计思路



仿照软件设计中的 mvc 架构，我们将工程设计分为三层：

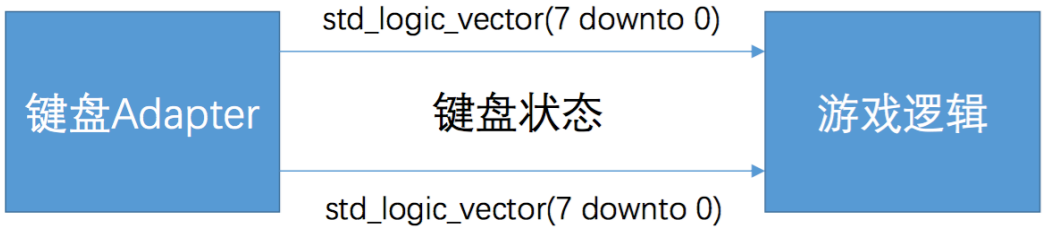
- 输入输出层，主要有 VGA Adapter 和 Keyboard Adapter
- 颜色控制层，有颜色计算模块和 sram 读取模块

■ 游戏逻辑层，庞大的逻辑计算模块

相互之间的接口如下：

输入输出层将玩家键盘操作传输给逻辑：

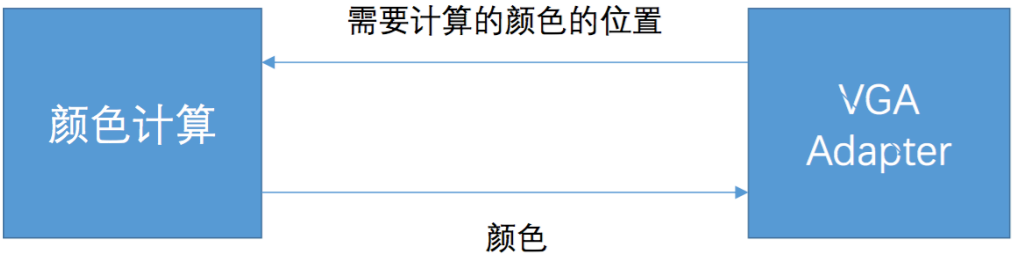
这里使用了两个 8 位 std_logic_vector，表示某玩家对应的操作键是否按下。



逻辑将计算好的显示信息交给颜色计算模块：

这两部分高度耦合，我们将二者写入同一个 logic 实体中。共用表示物体对应位置、物体对应图片编号、物体朝向等信息的信号。

颜色处理模块根据 sram 中储存的图片，计算出覆盖情况（处理透明）等，等待 vga adapter 询问时输出。



顶层设计：

只需例化出 keyboard Adapter、logic、vga adapter 各一个，并且对接好上述相应接口。

三、关键技术分析

1. sram 读取

sram 读取必须满足时序要求，先设置模式为读取，每次给定一个 21 位的地址，其中最高两位为 0，以 25MHz 的频率在一个周期之后读取上一个周期给定的地址的值，这里需要延迟计算，并不像软件代码直接获取数据。

2．颜色存储

sram 一个地址对应了 32bit 的数据，采用 8bit 保存一个像素颜色的存储方式，稍微牺牲准确度，大幅提高了 2M 内存的利用率，完整保存下了所有素材（1.92M），另外还降低了组合逻辑的延迟，因为四个像素代码中是乘除 4。

具体来说，rgb 颜色本来需要各 3bit，加上透明需要 1bit，舍弃蓝色最低位并用特殊颜色代替透明就可以用 8bit 保存一个像素颜色。

对于素材图片的存储，先按图片编号顺序再按行四个像素一组依次写入 sram 地址对应的 32bit 中，记录下每一福素材图片起始像素所在地址，这样将图片从二维转成类似一维的二进制数据，高效利用了 sram 的空间，同时如果知道图片编号和需要的像素位置，可以直接计算出它颜色信息在 sram 中存放的地址。

3．一次处理 40 个像素颜色

由于 sram 读取时序限制，需要 25MHz 的频率才能稳定过一个周期读取，为了保证屏幕刷新速度，必须至少以 25MHz 速度得到像素颜色信息，直接计算一个像素颜色将无法有效处理图片重叠的情况（另一种做法是用片内 rom 存储透明的情况，只用读取一次 sram 值，本项目素材太多无法按此方法实现）。每次只做一个像素颜色，会浪费每次读取的另外三个像素颜色数据，可以利用这一点一次提前处理 40 个像素颜色，利用好一次能读取 4 个像素颜色的特性。

由于素材图片存放的顺序是按行存储，屏幕像素颜色也是按行给出，而一行屏幕是 640 个像素，这样保证了一次处理的 40 个像素位于屏幕的一行中，同时对于会显示在这 40 个像素中的图片，需要给出的像素颜色必然是连续地存放在 sram 中的，下面以一幅图片的 12 个像素为例：

可以看出，最坏情况下这 12 个像素颜色会存储于 4 个 sram 地址中，相应的 40 个像素至多需要访问 11 次 sram 的地址，而一次处理 40 个像素意味着有 40 个周期可以访问 sram，可以有效处理三个图片重叠的情况，对于这个拳皇游戏来说已经基本够用了。

四、下载验证方法

图片的下载：使用 RLab 把二进制数据文件 data 下载到 SRAM。

使用 Quartus 13.0 打开 king.qpf 文件，直接编译然后下载，由于时序还存在些问题不保证每次下载都正常，可能要多次下载。

演示说明

输入输出使用 PS/2 键盘和 VGA 显示器。

进入开始界面后按回车进入游戏界面

P1 的方向控制为 WASD，拳为 H，踢为 J，防御为 K

P2 的方向控制为小键盘上下左右，拳为小键盘 1，踢为小键盘 2，防御为小键盘 3

五、实验中遇到的问题及解决方法

1. ROM 空间不足，无法保存所有素材。采用将图片转成二进制的形式通过 rlab 写入 sram 然后再读取解决空间问题。
2. 单独按顺序计算像素颜色时序不合法。改成一次处理 40 个像素颜色解决。
3. 一开始一个地址三个像素，获取像素颜色需要乘除 3 的操作。改成一个地址四个像素既解决了时间问题也解决了空间问题。
4. 和逻辑整合后时序依然存在问题。将图片行长度改成了 2 的幂次，减少了像素坐标变成一维时乘法的复杂度，另外将部分组合逻辑的变量改成了信号分周期做以减少时序要求，部分解决了时序的问题。