## 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。

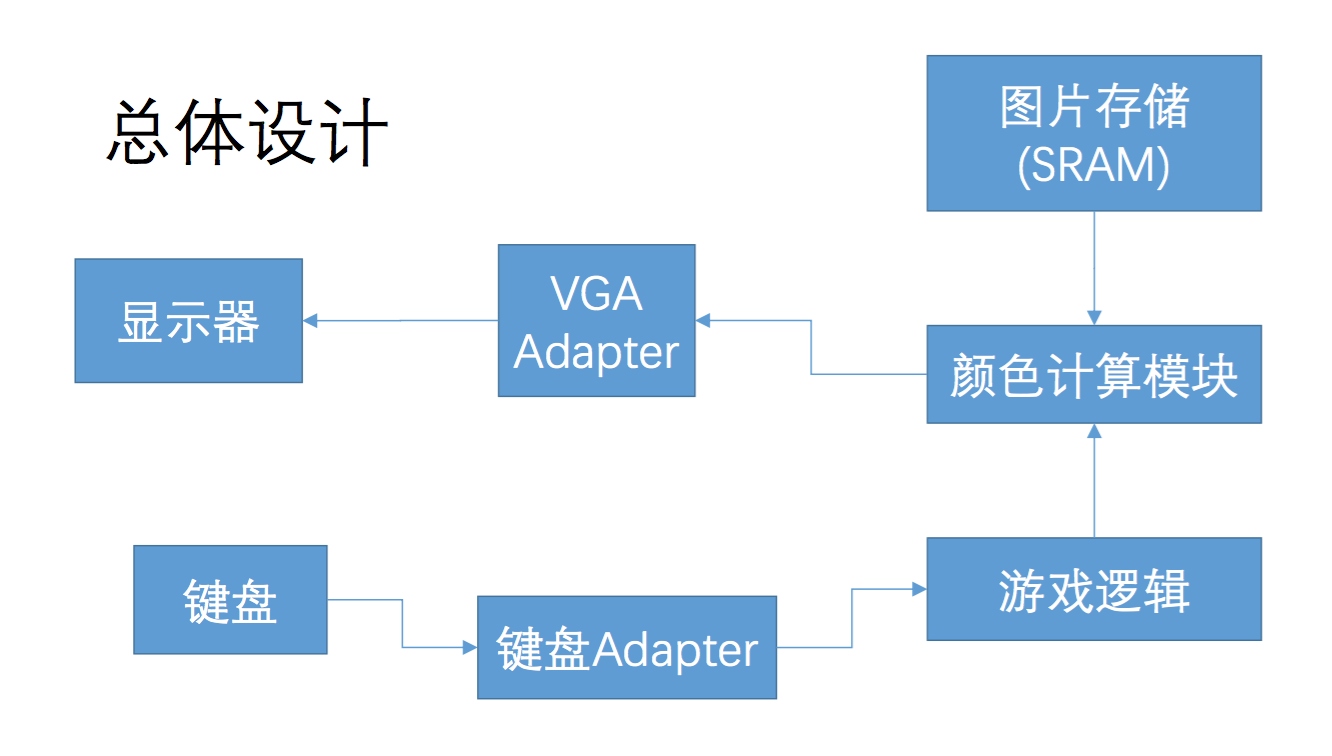
在项目根目录下包括以下一些文件和文件夹：

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



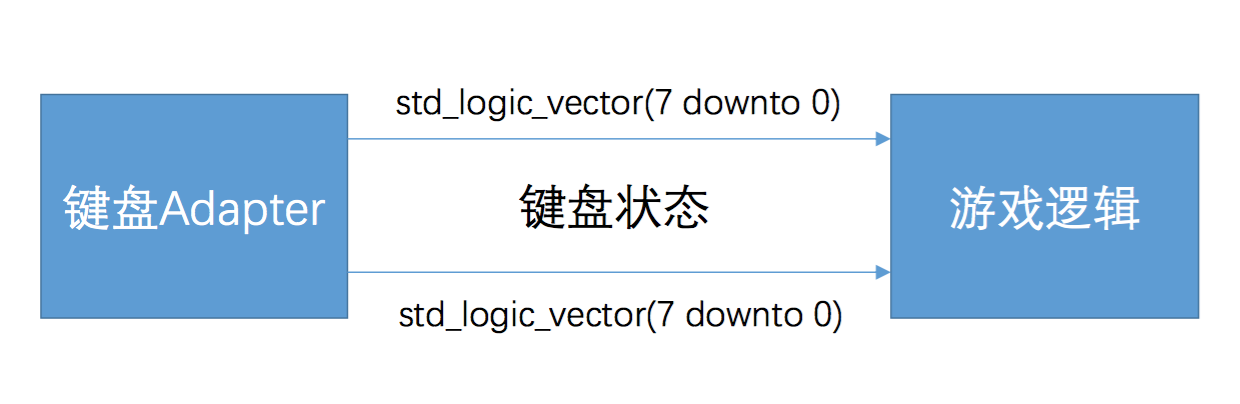
仿照软件设计中的mvc架构，我们将工程设计分为三层：

* + 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：

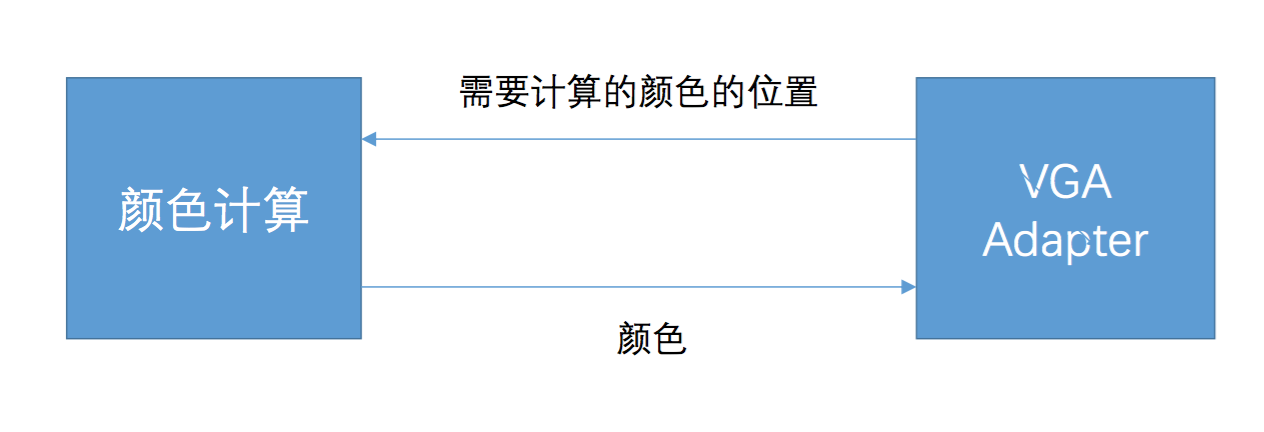
这里使用了两个8位std\_logic\_vector，表示某玩家对应的操作键是否按下。



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。

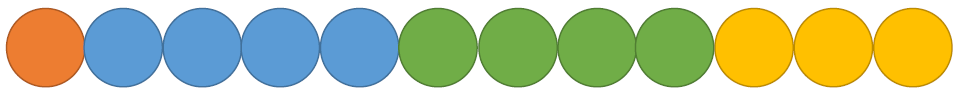
具体来说，rgb颜色本来需要各3bit，加上透明需要1bit，舍弃蓝色最低位并用特殊颜色代替透明就可以用8bit保存一个像素颜色。

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。