基于 Genesys 2 俄罗斯方块游戏设计与实现

王一土 16300200010

概要

本项目——基于 Genesys 2 FPGA 开发板的俄罗斯方块游戏设计与实现，使用 TOP DOWN 设计流程，完成了系统设计、RTL 设计、综合、功能和时序仿真、布局布线并且于 XILIN 公司 K7 系列的 Genesys 2 开发板上实现。游戏界面采用 VGA（640*480）显示。游戏实现了对俄罗斯方块左右移动，顺时针旋转以及当前方块与置换区方块交换等功能。（**游戏实现 demo 视频可见附件**）

附件链接：https://pan.baidu.com/s/1v8SBSFf-VQs8OBgsYAcksQ

提取码：d9ro

一、 设计规划

1. 设计要求

以复旦微电子公司 JFM4VSX55 DEMO 板为硬件实现平台，设计一个俄罗斯方块的游戏。因目前手上无此 FPGA 开发板，故先用 Genesys 2 代替。之后若可获得 JFM4VSX55 DEMO 板，可重新实现。

2. 设计思路

对此俄罗斯游戏的设计与实现，需要根据具体的游戏规则要求设计系统框图，RTL 实现各个子模块的逻辑功能（比如对方块的移动、旋转和置换，对游戏的积分以及判断游戏结束等），以 VGA 显示游戏界面。同时将游戏界面需要显示的静态数据（图片）存入 ROM 以方便读取。ROM 中的数据以 COE 文件格式存放。图片文件可以 python 调用 opencv 处理，COE 文件可用 matlab 写相关脚本文件生成。

故本项目的工作大致可以分为三个部分：

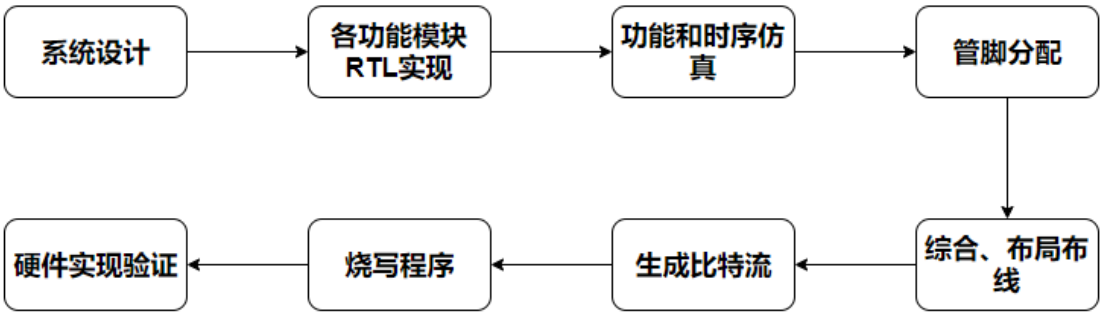
A．根据游戏规则，RTL 实现俄罗斯方块游戏基本功能逻辑模块并进行功能和时序仿真

B．根据 VGA 输出原理，RTL 实现 VGA 显示模块。（分辨率为 640*480 像素）

C．根据游戏界面设计，对显示数据（图片）进行处理并以相关脚本生成 COE 文件(py 与 matlab 文件见附件)

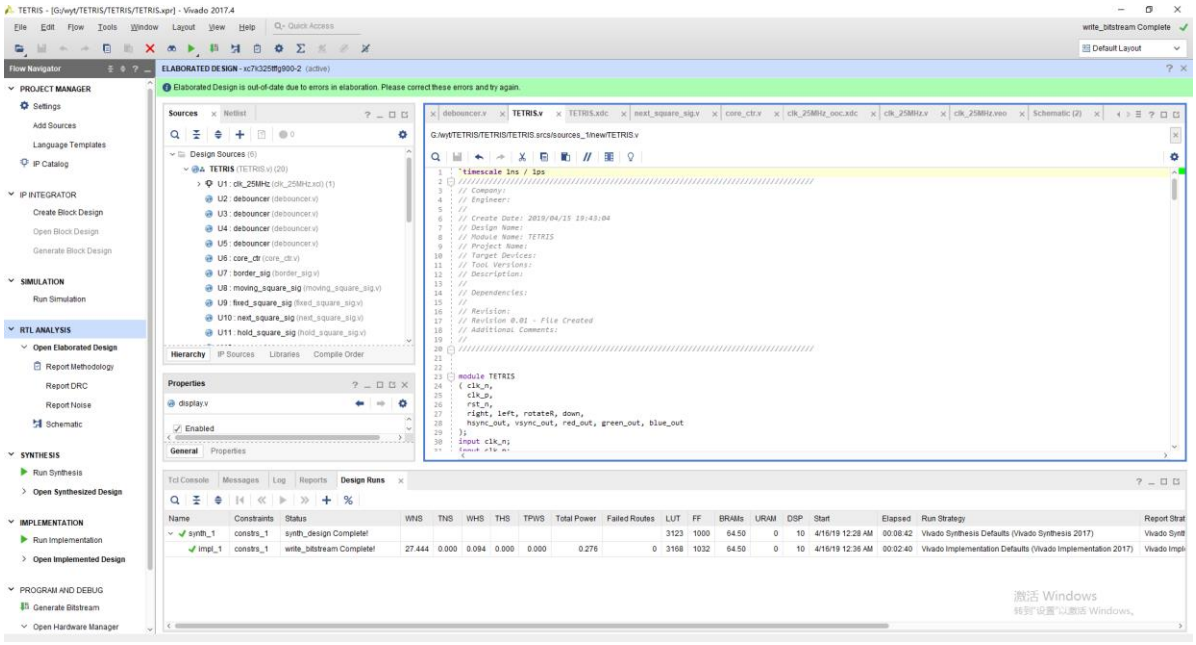
3. 设计流程及 EDA 使用

A. 设计流程图



B. EDA 使用说明

本项目使用的 EDA 工具为 Vivado 2017.4。下图为此软件的开发界面。



Vivado 2017.4 开发界面

利用 Vivado 2017.4 可进行 RTL 实现、功能和时序仿真、管脚分配、综合、布局布线、生成比特流、烧写程序等步骤。以下为 Vivado 2017.4 提供的 Project Summary。

Settings Edit

Project name:	TETRIS
Project location:	G:/wyt/TETRIS/TETRIS
Product family:	Kintex-7
Project part:	Genesys2 (xc7k325tffg900-2)
Top module name:	TETRIS
Target language:	Verilog
Simulator language:	Mixed

Board Part

Display name:	Genesys2
Board part name:	digilentinc.com:genesys2:part0:1.1
Connectors:	
Repository path:	G:/sth/viviado_install/Vivado/2017.4/data/boards/board_files
URL:	www.digilentinc.com/genesys2
Board overview:	Genesys2

Synthesis

Status:	Complete
Messages:	152 warnings
Part:	xc7k325tffg900-2
Strategy:	Vivado Synthesis Defaults
Report Strategy:	Vivado Synthesis Default Reports

Implementation Summary | Route Status

Status:	Complete
Messages:	91 warnings
Part:	xc7k325tffg900-2
Strategy:	Vivado Implementation Defaults
Report Strategy:	Vivado Implementation Default Reports
Incremental compile:	None

DRC Violations

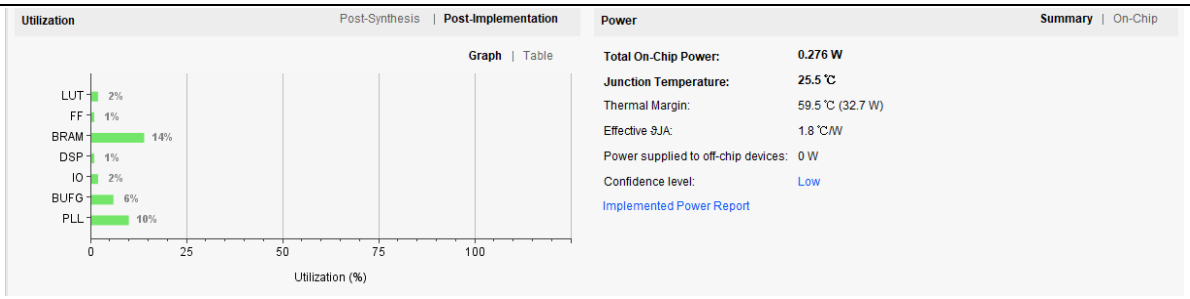
Summary: 56 warnings

Implemented DRC Report

Timing Setup | Hold | Pulse Width

Worst Negative Slack (WNS):	27.444 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	2372

Implemented Timing Report

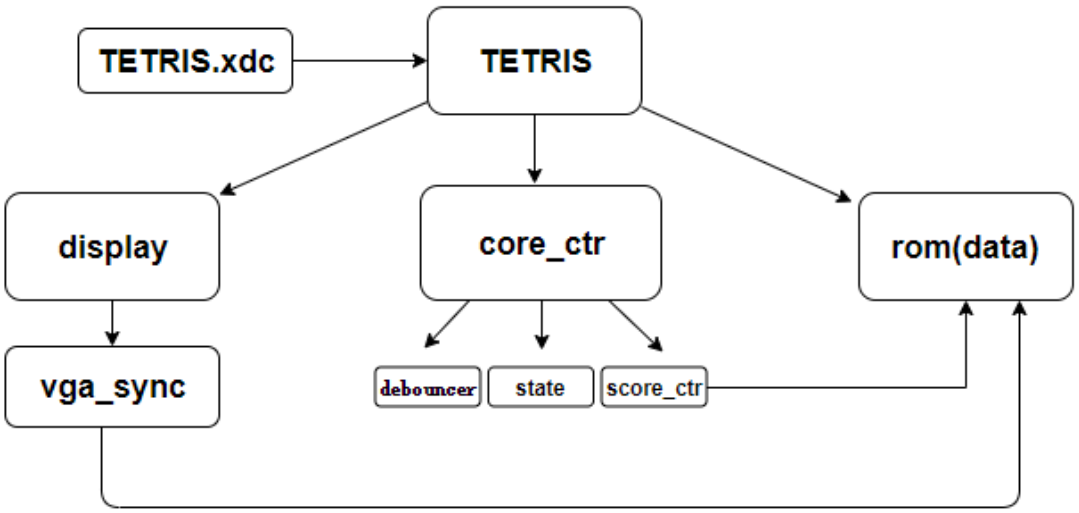


Project Summary

关于 Vivado 的使用详细介绍，可参阅： 《Xilinx Vivado 的使用详细介绍（2）：综合、实现、分配管脚、时钟设置、烧写》
https://blog.csdn.net/jzj1993/article/details/45533769

二、 设计实现

1. 框图介绍



系统框图

TETRIS 为顶层模块，规定了输入输出以及内部各子模块的链接；
display 模块负责以 vga 信号显示游戏信息，同时以 vga_sync 模块控制 vga 接口的行同、场同步以及红绿蓝信号；
core_ctr 模块为核心控制模块，主要是逻辑实现游戏功能，其中还包括 debouncer 防抖动模块、state 状态机以及 score_ctr 计分模块；
rom 模块其实由 5 各 rom 组成，分别显示游戏界面中"NEXT""HOLD""GOAL""OVER!"字符以及计分模块输出的当前得分数字；

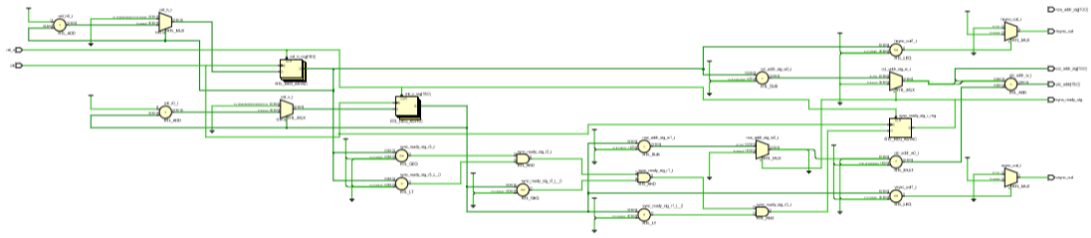
- 游戏设计界面如下：（大致设计，进行配色之前，尺寸为 640*480 pixel）

各子模块设计与验证详见下文。

2. 各模块/IP 核设计与验证 综合与实现

A. MODULE – vga_sync

a) RTL 设计实现



RTL Schematic

对 vga 信号的控制模块是一个相对独立的模块。此模块主要是输出对 vga 接口的场同步与行同步；同时根据 vga 的具体时序，计算出目前显示的是屏幕上的像素的位置 col_addr_sig（列坐标）以及 row_addr_sig(行坐标)。而 pic_addr 则是讲二维坐标一维化, pic_addr = row_addr_sig*640+col_addr_sig，以此方便读取 rom 中存储的图片信息，sync_ready_sig 是表示当前扫描的点是否可以输出显示。

（**本项目采用 640*480 像素，故图片的尺寸为 640*480**）

RTL 实现此模块的关键在于理解 vga 接口的原理。下面就 vga 接口的原理进行简单介绍。

1) 时序讲解

扫描从屏幕左上角开始，从左向右逐点扫描；

每扫描完一行，电子束回到下一行的起始位置，期间 CRT 对电子束进行消隐；

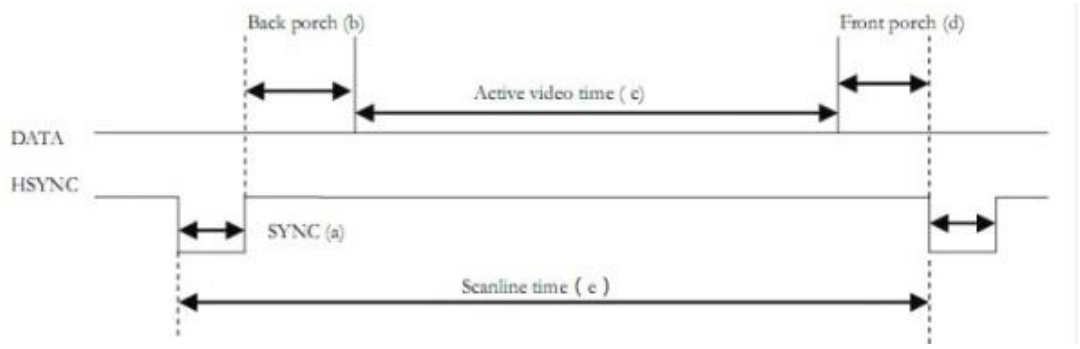
每行结束时，用行同步信号进行同步，当扫描完全部的行，形成一帧，用场同步信号进行同步，回到屏幕的左上角，同时进行场消隐，开始下一帧；

完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂

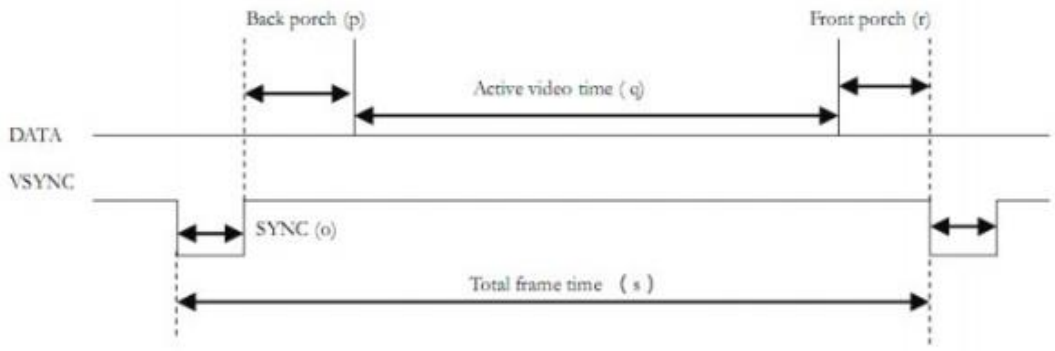
直扫描时间，其倒数称为场频率，即刷新一屏的频率，常见的有 60Hz，75Hz 等等。标准的 VGA 显示的场频 60Hz,行频 31.5KHz;

行场消隐信号：电子枪所发出的电子束从屏幕的左上角开始向右扫描，一行扫完需将电子束从右边移回到左边以便扫描第二行。在移动期间就必须有一个信号加到电路上，使得电子束不能发出。不然这个回扫线会破坏屏幕图像的。这个阻止回扫线产生的信号就叫作消隐信号,场信号的消隐也是一个道理;

时钟频率：以 640x480@59.94Hz(60Hz)为例，每场对应 525 个行周期(525=10+2+480+33),其中 480 为显示行。每场有场同步信号,该脉冲宽度为 2 个行周期的负脉冲，每显示行包括 800 点时钟,其中 640 点为有效显示区,每一行有一个行同步信号，该脉冲宽度为 96 个点时钟。由此可知：行频为 525*59.94=31469Hz,需要点时钟频率：525*800*59.94 约 25MHz;（**本项目采用的就是此时钟频率**）



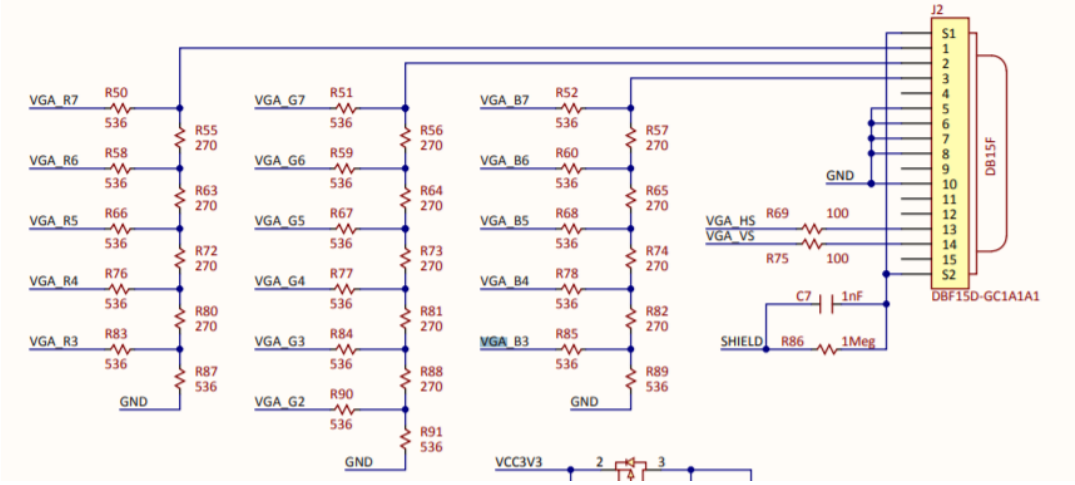
VGA 行时序



VGA 场时序

2) Genesys 2 VGA Schematic

VGA 接口通常有 15 个输入输出信号，但是不同 FPGA 开发板的 VGA 接口的信号略有不同，以下为本项目所用 Gensys 2 的 VGA Schematic。



Genesys 2 VGA Schematic

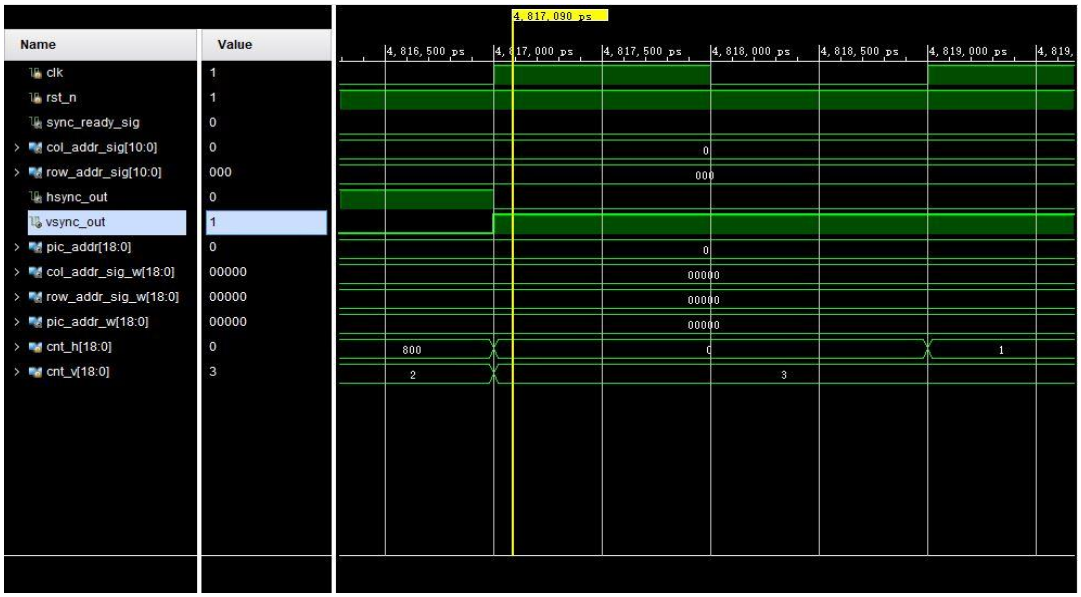
可见，此 VGA 接口由 VGA_R[4:0]， VGA_G[5:0], VGA_B[4:0], VGA_HS, VGA_VS 构成。

b) 验证与仿真

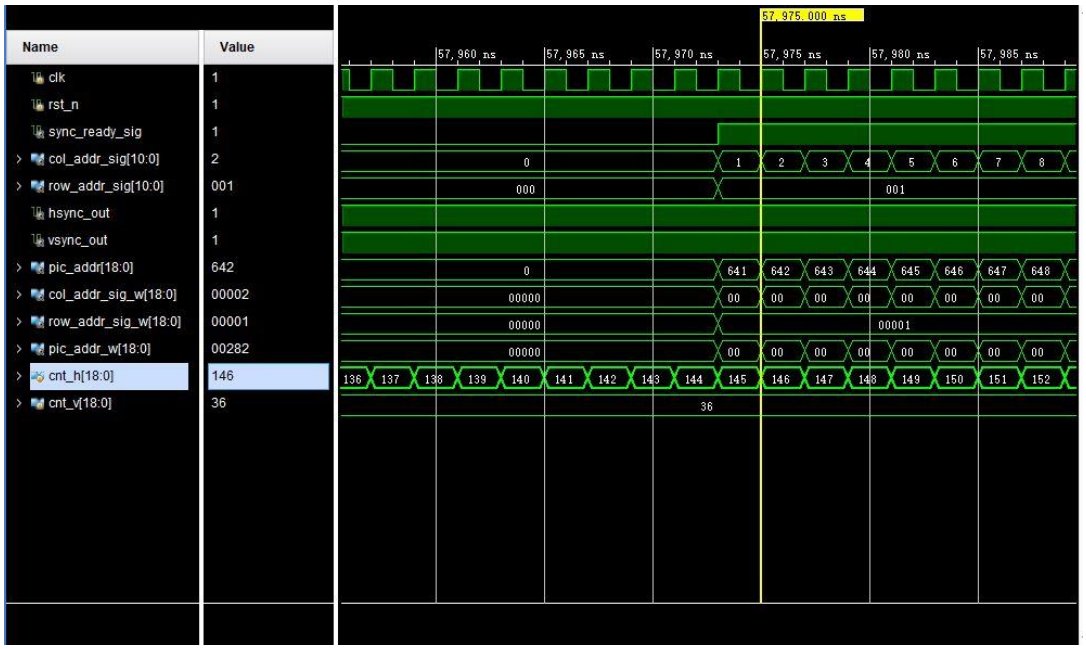
此模块的仿真激励为 clk 和 reset_n，主要验证的是行同步与场同步信号是否处在规定的时序，列坐标与行坐标对应时序的变化是否正确以及图像一维坐标的计算结果。具体仿真波形如下。

1) 行同步 hsync_out

2) 场同步 vsync_out

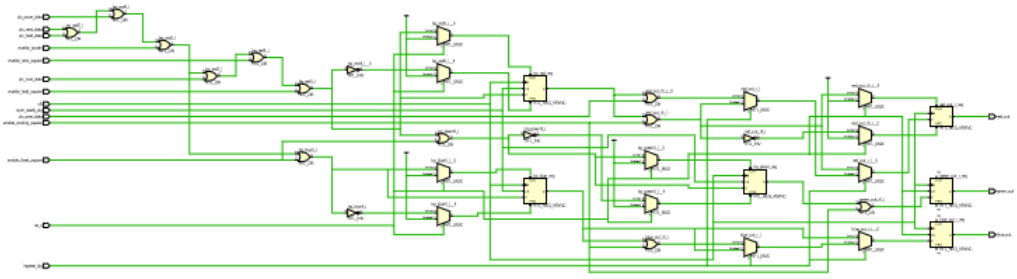


3) 列坐标 col_addr_sig 行坐标 row_addr_sig 一维图像地址 pic_addr



B. MODULE – display

a) RTL 设计实现



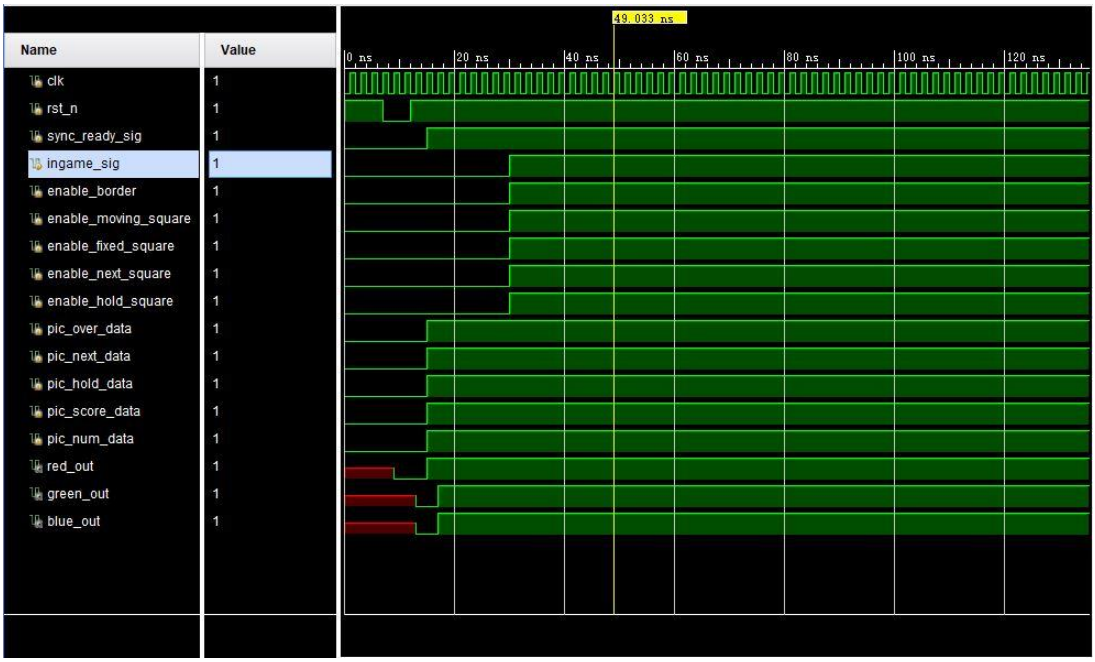
RTL Schematic

此模块的主要功能为为 vga 接口提供 RGB 信号。提供 RGB 输出信号的原理为：**当 VGA 扫描到相应的位置时，此模块接收到 vga_sync 模块提供的 sync_ready_sig 为真的信号，表示当前扫描到的像素可以输出。而输出的内容则由存储的数据图片以及方块在此像素点是否有显示来决定；而显示的颜色则由 RGB 信号的搭配来决定**。为简便配色步骤，本项目中只使用 RGB 信号各 1bit。核心代码段如下：

```
bg_red   <= pic_next_data | pic_hold_data | pic_score_data | enable_border | pic_num_data | enable_next_square | enable_hold_square;
bg_green <= pic_next_data | pic_hold_data | pic_score_data | enable_border | pic_num_data | enable_next_square | enable_hold_square | enable_fixed_square;
bg_blue  <= pic_next_data | pic_hold_data | pic_score_data | enable_border | enable_fixed_square;
```

b) 验证与仿真

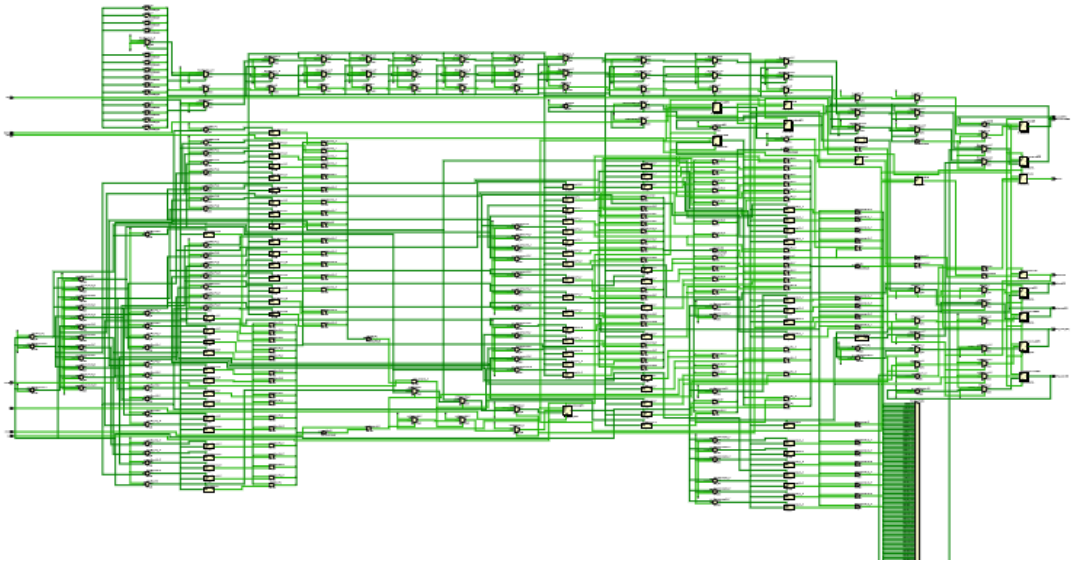
输入激励就为相关要显示内容对当前像素点的“决定”，依此检验当前像素点应该显示的 RGB 信号。



根据上述核心代码的“配色”，可以得到如下实际游戏界面图像。

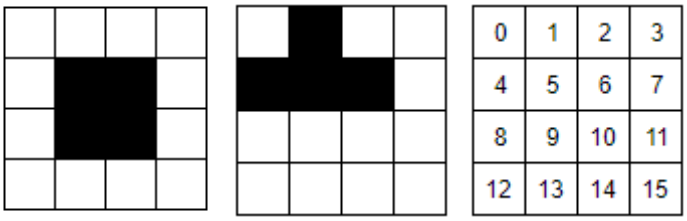
C. MODULE – core_ctr

1) RTL 设计实现



RTL Schematic

此模块为游戏的核心控制模块，绝大部分游戏涉及的逻辑功能都在此实现。
首先要说明的是 7 种类型的俄罗斯方块都以 4 格*4 格来管理实现：



如上图所示；并且对这 16 个格子进行从 0 至 15 的编号，其中 0 标志位为此俄罗斯方块的位置标志，也就是代码种的 moving_square_loc。而每一个方块类型也都有自己的编码，例如左一的方块类型的编码为 0000_0110_0110_0000，1 表示该位置有方格填充，0 则反之。
此模块种有一个尤为重要的寄存器 fixed_square_map，此寄存器的作用是来保存已经落下并且未被消除的方块的位置信息以及边界信息。
方块的堆叠区，也就是上文游戏界面设计的 GAME DISTRICT 的大小，以一个格子的大小来量度，是 14*10；以 pixel 来量度，就是 280*200pixel。
上述约定主要是为 RTL 实现各逻辑功能提供方便。实现的各功能如下：

a) 每一秒下降一格；（根据游戏界面设计，一格的边长为 20pixel；系统时钟为 25MHz）
b) 判断能否下落并且决定是否下落；（即判断方块下方是否已经有方块）
c) 判断是否可以右移并且进行或不进行右移；（即判断右移之后是否超出游戏区）
d) 判断是否可以左移并且进行或不进行左移；（即判断左移之后是否超出游戏区）
e) 判断是否可以与置换区方块置换并且进行或不进行置换；（即判断置换之后是否与之前落下并未被消除的方块有位置重叠）
f) 判断是否可以旋转；（即判断旋转之后是否与之前落下并未被消除的方块有位置重叠）
g) 判断是否成功相撞；（即判断是否与下一行的固定方块是否相衔接）
h) 顺时针旋转 90 度操作；（根据约定的位置编号，进行对应位置的方格交换）
i) 产生下一个方块类型；（通过产生不同的编码，产生不同的方块类型）
j) 判断是否可以消去一行；（判断每一行是否全都被方格填满）
k) 固定方格位置与显示的更新；（根据消去的结果，对寄存器 fixed_map_square 进行更新）
l) 对消去的行数进行计数；（对应玩家的得分，为之后的模块提供输入）
m) 判断游戏是否结束；（即判断堆叠的方块最大高度是否超出了游戏区）

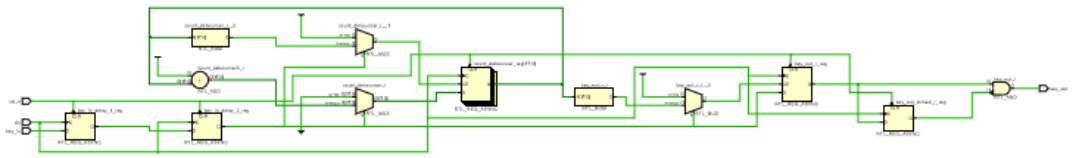
此模块的输入输出都是基于以上操作，详细代码可见附件工程中的代码。

2) 验证与仿真

此模块过于庞大与繁杂，主要功能的验证也是很繁琐，故此处不再附仿真波形（太多了）。经过软件与硬件仿真，以上功能均能得到满足，可见附件中视频的 demo。

D. MODULE – debouncer

1) RTL 设计实现

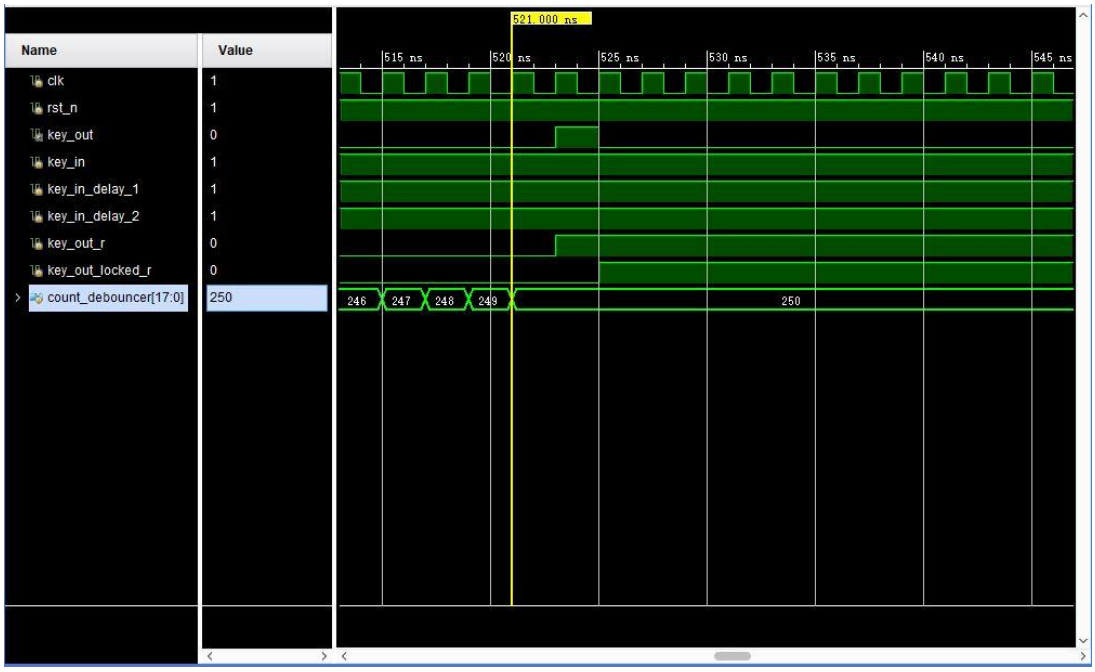


RTL Schematic

此模块的功能为为输入按键提供防抖动功能。主要原理是为按键提供一个 10ms 的 delay，即连续检测到 10ms 的输入信号才产生一个周期的输入脉冲。此处要注意的是，不同开发板的 push button 按下的高低电压不同，Genesys 2 开发板的 push button 按下为高电平，置空为低电平。

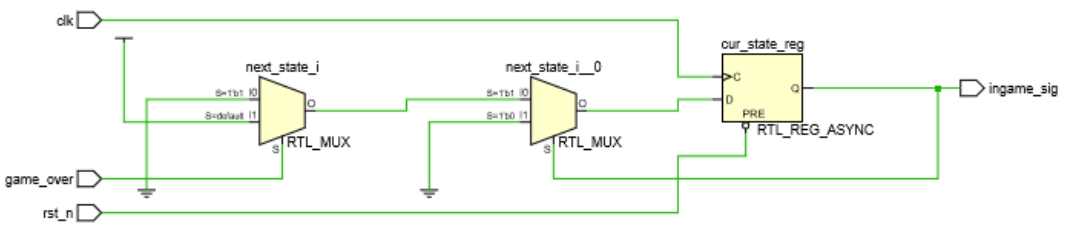
2) 验证与实现

由于系统时钟设置为 25MHz,10ms 为 250000 个时钟周期，仿真时不必这样，所以仿真时我把 RTL 改为连续检测到 250 个周期的输入就可以。仿真波形如下。



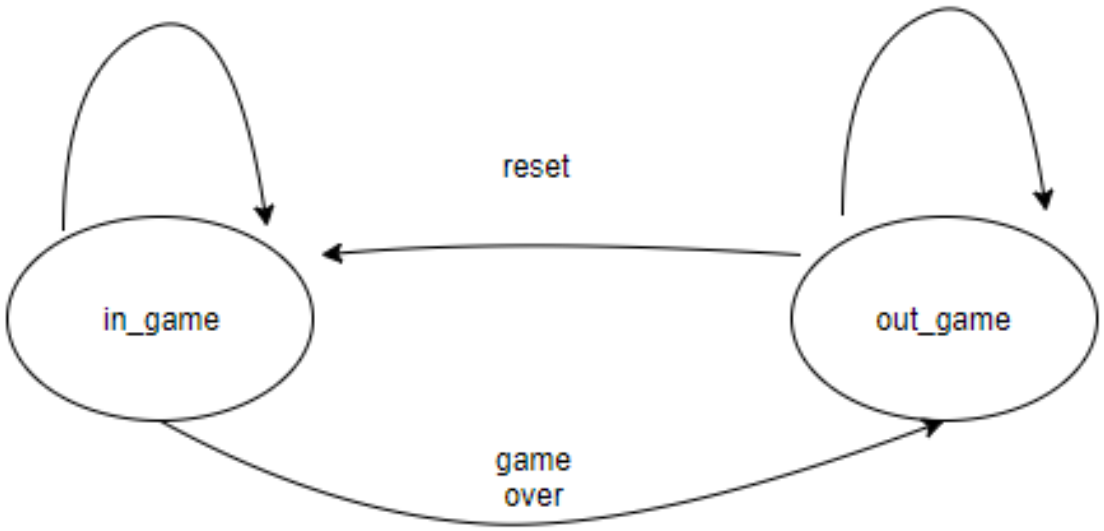
E. MODULE – state

1) RTL 设计实现



RTL Schematic

此模块为状态机，状态只有 in_game 和 out_game 两个，很简单但又非常有用，其他模块的很多信号都受此控制。状态跳转图如下。

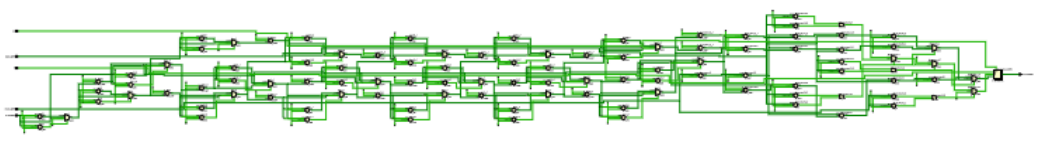


2) 验证与仿真

主要检验状态是否能正确跳转，仿真波形如下。

F. MODULE – score_ctr

1) RTL 设计实现



RTL Schematic

此模块为计分模块，对 core_ctr 模块输出的消去行数的信号进行接收。此模块的设计原理为：将消去的行数进行从二进制到十进制的转换，得到十进制的分数，分别得到百位、十位以及个位的数字。

然后由此换算成 rom 中存储的地址对存在 rom 里的数字图像信息进行索引，由之前 display 模块的控制信息，此模块中决定在屏幕的相应位置是否有分数的显示，并从 rom 里读取数据。Rom 中存放数据的格式与地址在之后的模块中介绍。

其中数字的显示区域为： col：195－270 row：283-323 （这里还是把整个显示界面想象成一个**宽 640，高 480 的二维地图**）

2) 验证与仿真

主要验证二进制转十进制是否正确以及地址的转换是否正确。仿真波形如下。

G. MODULE – next_square_sig hold_square_sig moving_square_sig fixed_square_sig border_sig

1) RTL 设计实现

此四个模块大同小异，主要功能是为游戏的 NEXT 区、HOLD 区、固定的方块位置以及正在移动的方块提供显示信息。其中 NEXT 区、HOLD 区的方块位置是十分固定的，即以 4 格*4 格来管理的方块位置在游戏设计界面的位置是固定的。这里还是把整个显示界面想象成一个**宽 640，高 480 的二维地图**。

NEXT 区： col：191 - 270 row：51 – 130

HOLD 区： col：191 – 270 row：161-240

而对于 moving_square 和 fixed_square 则是要结合方块的移动信息以及固定方块的位置信息来提供显示信号。

2) 验证与仿真

这几个模块仿真也大同小异，主要是验证是否在规定的区域内显示了预期的信号即可。仿真波形太多，此处从略。

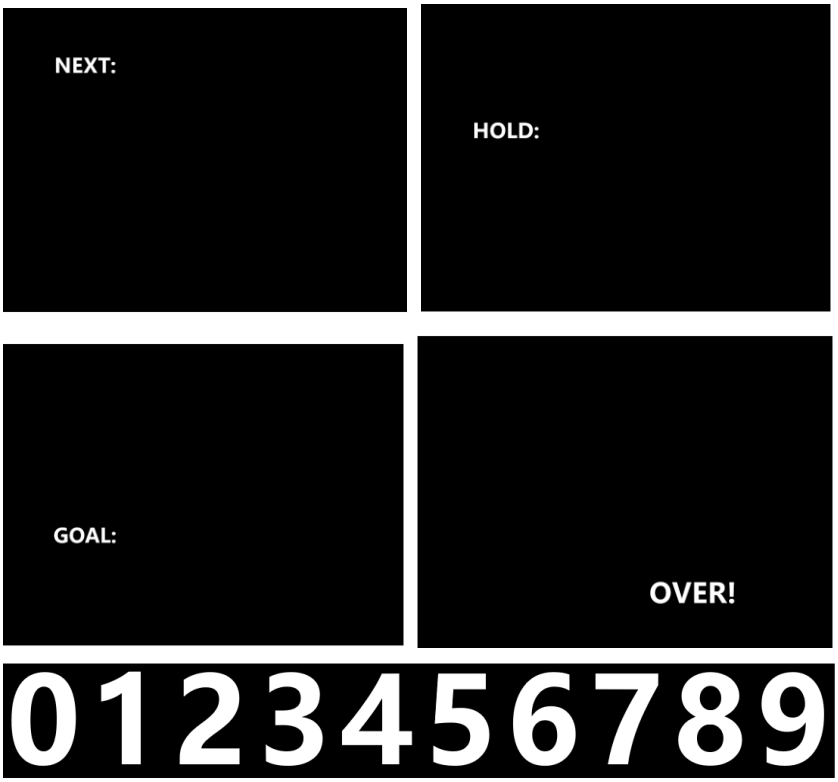
H. IP 核 – Block Memory Generator: pic_over pic_next pic_hold pic_score pic_num

1) 设计实现

工程中所用的 5 个 ROM 都是调用了 IP 核，他们分别用来存放游戏界面中的"NEXT:""HOLD:""GOAL:""OVER!"以及计分的数字。

调用 IP 以及生成 ROM 中的 COE 文件流程如下。

a) 分别生成图片，即要显示的图像轮廓；

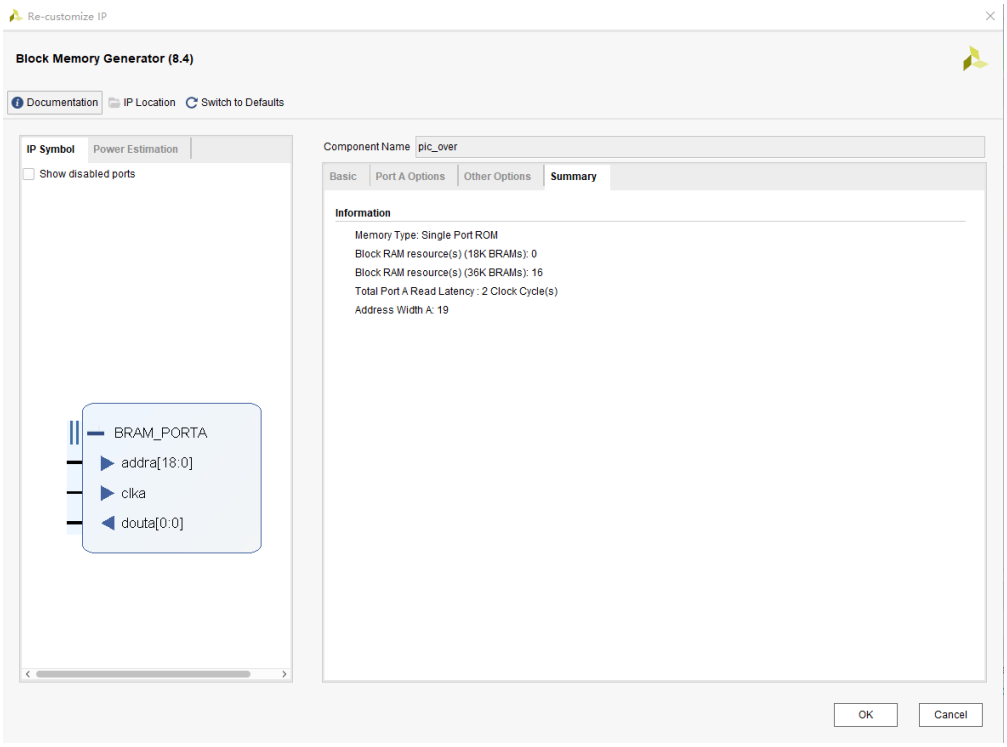


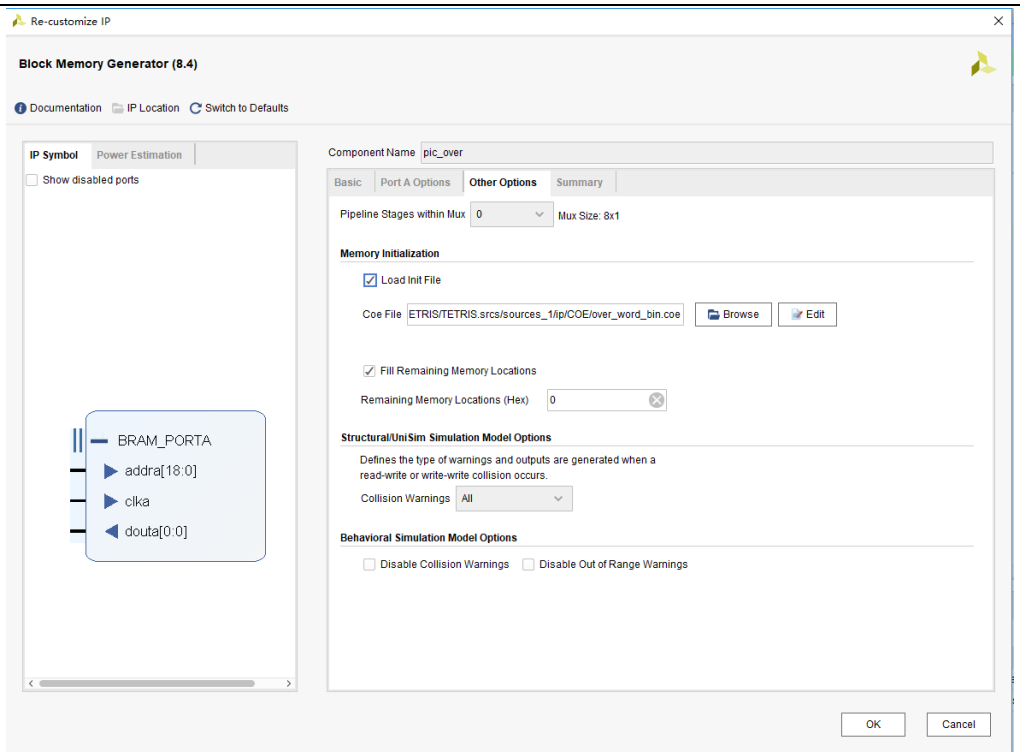
b) 用 Python 调用 opencv 把图像转换成 640*480 或者 250*40，并且二值化保存。
保存成此尺寸是为了与之前的代码中的寻址方式对应。
"NEXT:""HOLD:""GOAL:""OVER!"图片的数组长度为 640*480;
数字图片的数组长度为 25*10*40;

c) 根据 COE 文件格式的要求，用 matlab 写一个脚本，读取二值化的图像文件分别生成各自的 COE 文件。(信息就是上述的数组信息)

d) 调用 vivado block memory generator ip 核
相关代码脚本见附件。

2) 验证与仿真
生成的 IP 核信息如下:





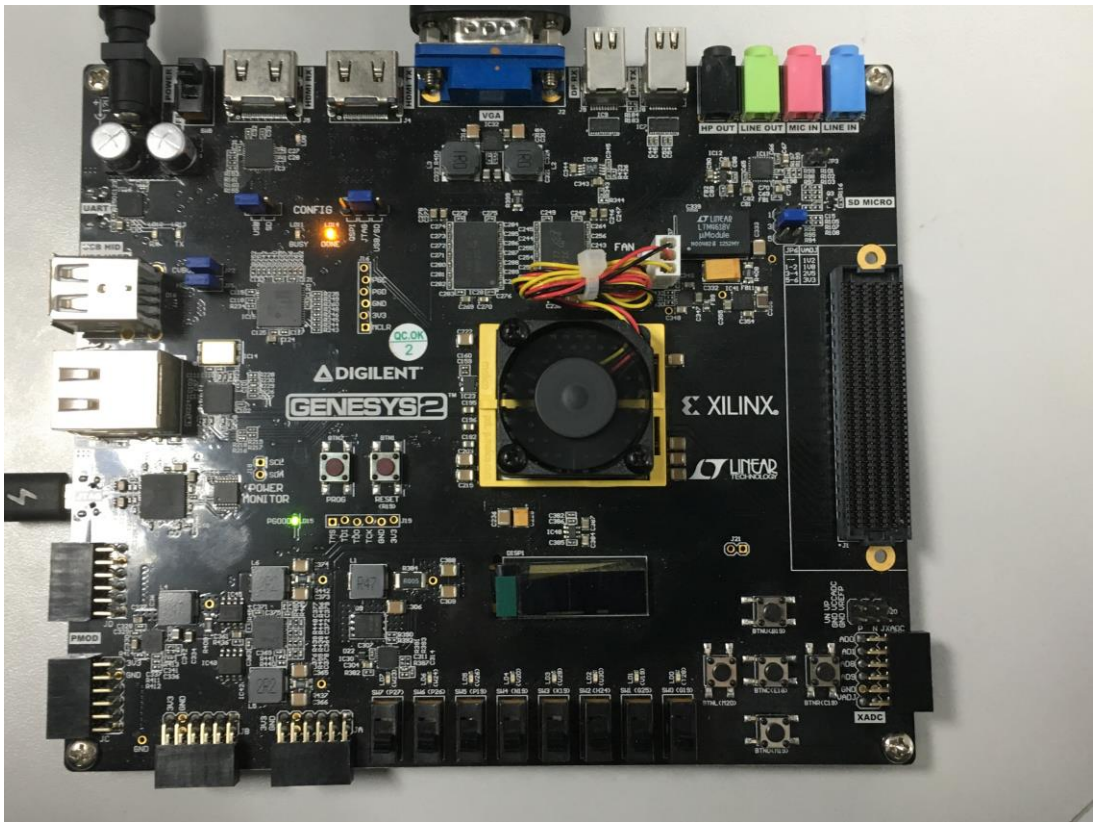
仿真读取 rom 数据如下。

3. 编程工具

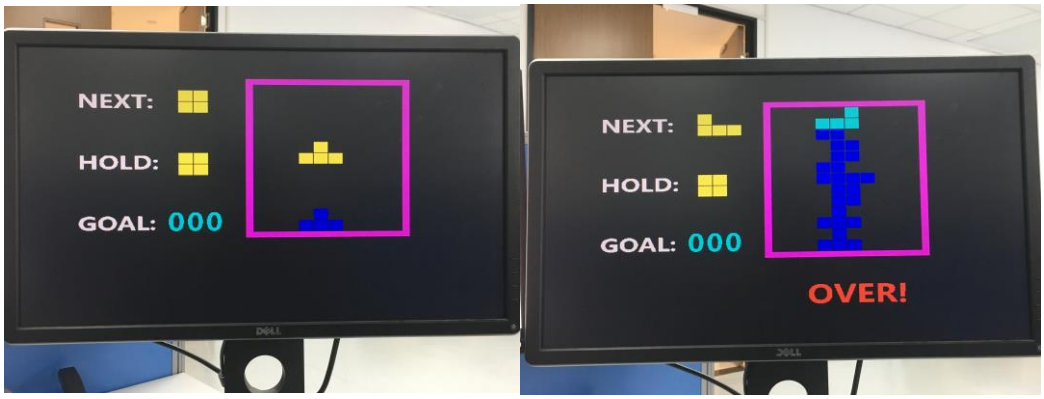
Verilog 及 Python 代码在 VScode 编辑器中完成，Verilog 代码在 Vivado 中编译调试，Python 代码直接在 VScode 中终端运行调试；
MATLAB 代码在 MATLAB R2016b 中完成调试运行。

4. 硬件测试成果展示

A. Genesys 2 开发板



B. 游戏界面



C. 视频演示

见附件视频 demo

三、 设计总结

1. 注意事项与编程技巧

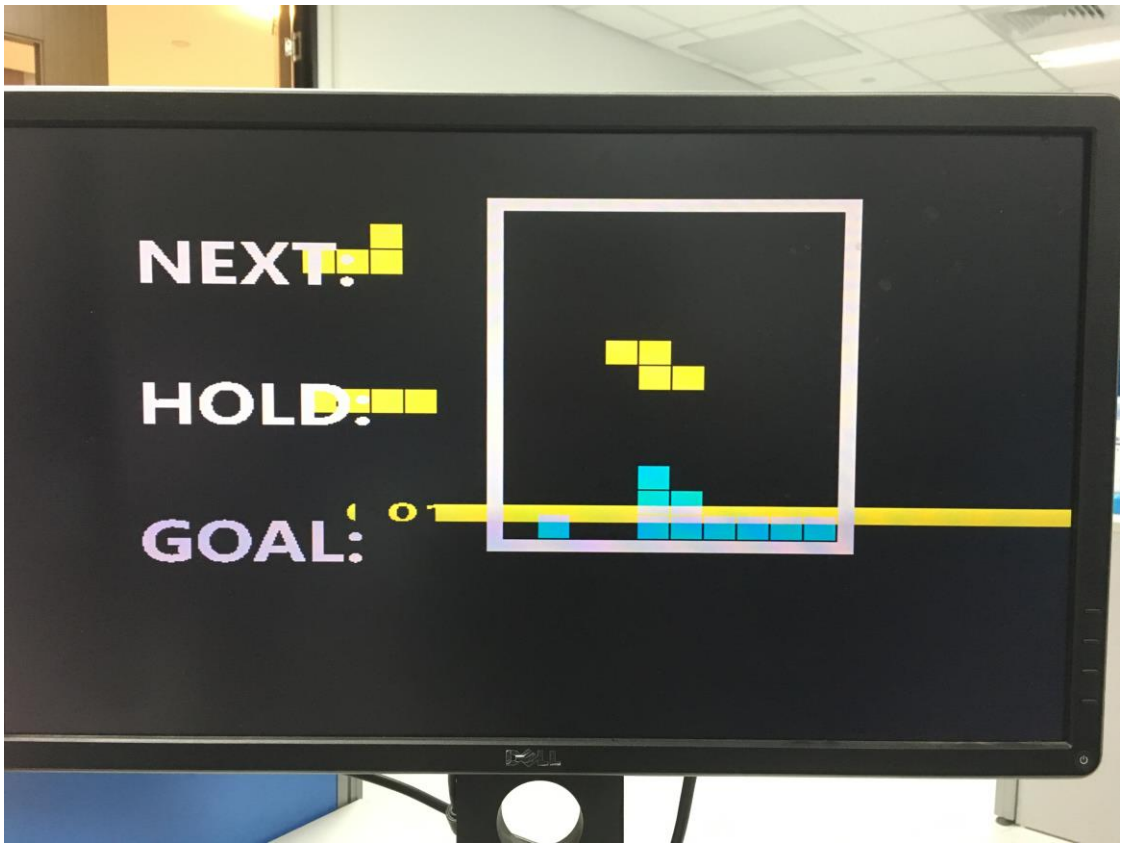
a) 此次写组合逻辑时运用了 always 结构，大大提高了效率，之前都是 assign 一路下来。但是要注意的是，always 结构中所有变量都要是 reg 类型。所以也由此看出，reg 变量不一定只能是用在时序电路中，在组合逻辑电路中也非常好用；

b) 在图片存储的时候，我都采用了将二维信息一维化存储，这样做是为了之后寻址的方便，只要计算准确即可。而且 ROM 中的数据我都是单 bit 存储，减小了带宽，也使结构变得简单。当然这样的缺点就是颜色搭配选择非常少，不过考虑到我也不是专门配色的，就无所谓了；

c) 同样是在确定某个二维像素点的位置的时候，可以采用分别取两个足够长的一维数组来采集行与列的信息，然后相与，来确定最后的信息。这一个技巧屡试不爽，有点难以形容，但是看了代码会显得很清晰易懂；

d) 关于对 ROM 存储信息的读取。当使用 ROM 的输出但是此时并没有访问 ROM 的时候，如果不事先约

定，你无法预测这个输出的值是多少。我在硬件仿真的时候就出现了这种情况，导致显示并非预期，有几行竟然全是“黄色”。 并且如果对图片的处理不够准确，导致相应的信息没有出现在相应的位置，显示的结果也会有偏差。如下是我调试过程中遇到的问题。



e) 在写约束文件分配管脚的时候，可以直接用 github 上 nice 的程序员提供的“模板”：https://github.com/Digilent/digilent-xdc/
里面有齐全的不同型号开发板对应的 xdc 约束文件模板，并且附有注释，大大提高了分配管脚的效率；

f) 关于本项目的时钟信号我调用了 clock wizard ip 核。由于 genesys 2 较为高端，其所用的时钟是差分时钟，一开始我只用了单端输入，并且由此写 xdc 文件约束管脚，结果却是布局布线错误。最后才发现这一问题，故由此提醒我在之后的工程中要注意所用开发板的具体情况。

g) 在处理移动的方块以及固定的方块的位置信息的时候，我也费了一些功夫。后来想出了可以先用一个寄存器从“宏观”山存储其位置，再用一个寄存器从 pixel 的量级来提供显示信息，问题从而得到解决。

2. 体会心得

通过这个 Project，我的 Verilog 编写能力以及 FPGA 开发能力再次得到了增强。由于之前也高过一个彩条旋转的 FPGA 实验，所以这次 Project 中，对于 VGA 显示的部分就显得比较得心应手。这几天做 PJ 的过程虽然有时比较枯燥，但是能力确实就是在这种实践项目中得到培养。相信之后自己对于 verilog 的编写以及 FPGA 的开发会有一个更高的认识。

当然，在这里也感谢 lab 里提供的 Genesys 2 开发板，以及朱浩哲学长对开发板使用上提供的工程指导。

3. 总结

本项目涉及面广，有 RTL 的电路实现，用 open cv 对图像的处理以及生成 coe 文件的脚本的编写，可以说是一个较为完整的工程项目。当做出来的东西真正 work 的时候，确实是比较爽。