浙江大學

数字逻辑设计基础Project实验报告

项目选题：关于小恐龙跳跃障碍及记分游戏的实现

课程名称：数字逻辑设计

姓名　：徐梓毓、杨辰

学号　：3220102605，3220105461

2024-01-07

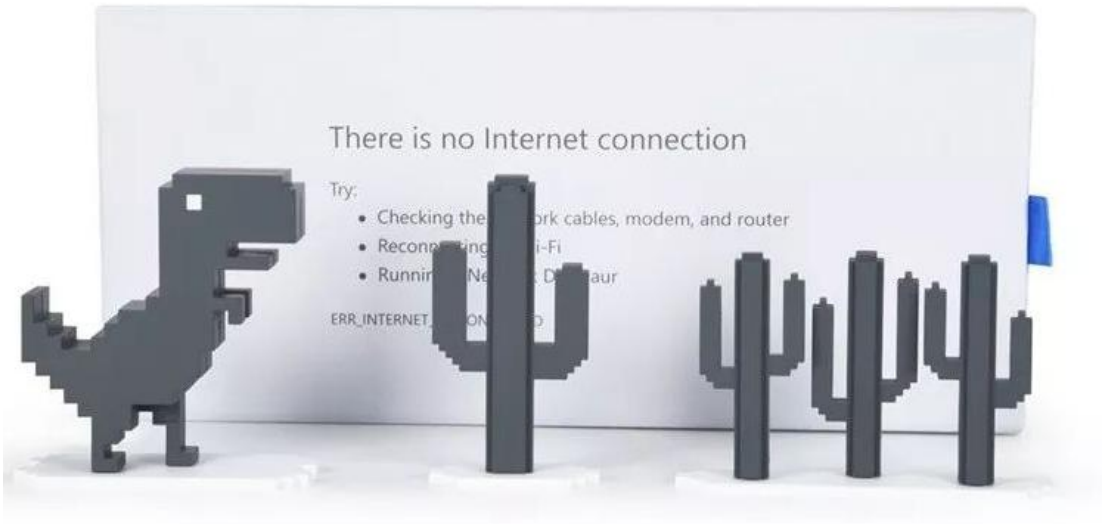
目录

实验背景介绍

在谷歌浏览器的小恐龙游戏的启发之下，我们想要实现一个类似的游戏。

游戏的基本原理是，在一个彩色的背景图中，有随机出现的云朵以及不断出现的障碍物，还有随机出现的果实，玩家将控制一只小恐龙进行闯关冒险。小恐龙奔跑的过程中，需要跳过障碍物。

玩家可以通过按键控制小恐龙的跳跃，走过的距离越远，分数越高。玩家控制小恐龙吃掉果实，可以获得额外的分数。玩家也可以设置不同的模式，简单模式将会保持一定的速度，难度模式将会随着时间的增加，速度越来越快。在记分板上可以看到自己的分数以及最高分数。



实验工具

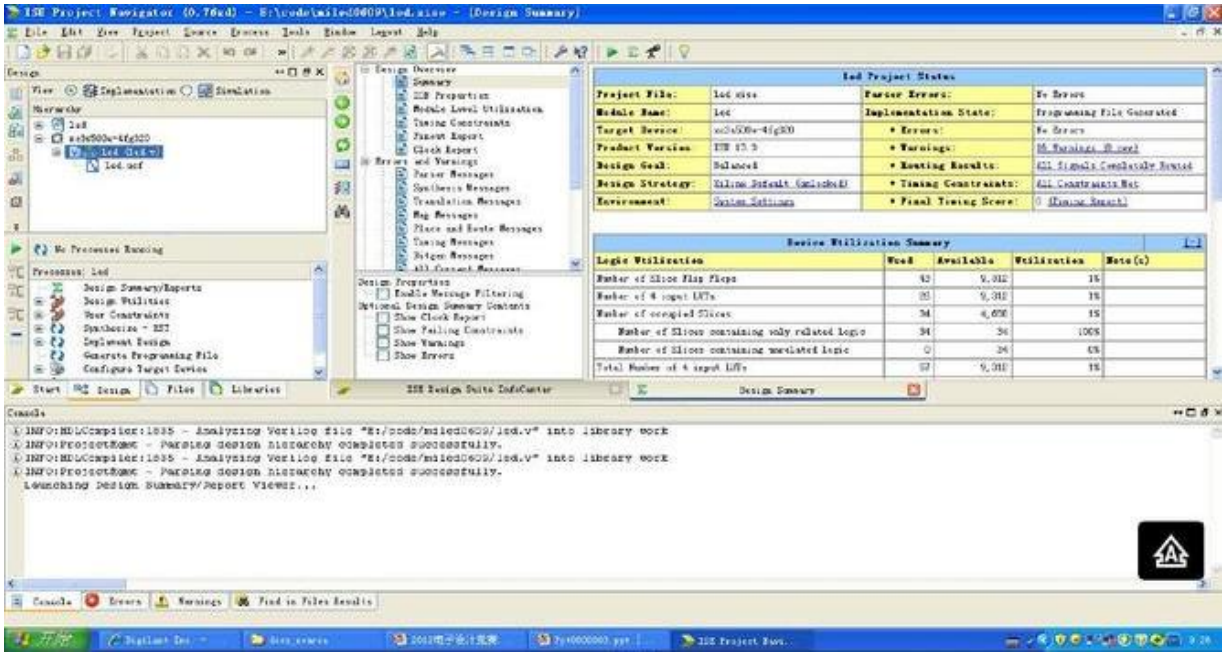
实验平台：　　Xlininx kitnex7

开发环境：　　Xlininx ISE 14.7

硬件描述语言：　　Verilog HDL

本次设计所用的硬件编程软件为 Xilinx 公司设计的 ISE 14.7 软件，所使用的语言为 Verilog 语言。该软件的使用和 Verilog 语言的学习都较为简单，和 c 语言有相似之处。同方式或混合方式对设计建模。

这些方式包括：行为描述方式建模；数据流方式建模；结构化方式建模等。Verilog HDL 中有两类数据类型：线网数据类型和寄存器数据类型。线网类型表示构件间的物理连线，而寄存器类型表示抽象的数据存储元件。通过模块化的设计与相应的外部接口，可以实现较好的人机交互。



实验原理

一、实验预备原理

1. COE 文件

COE 是"content Editor"的缩写，它是一种文本文件格式，其中包含了 FPGA 设计中的初始化数据。

COE 中第一行表示 COE 使用了 16 进制，

第二行开始后面均为 RGB 值。

```
memory_initialization_radix=16;
memory_initialization_vector =
```

我们使用的 COE 文件中每个数值均为二进制 12 位数，对应存入 COE 中泽对应 3 个十进制数字。大部分图片的储存方式都是三通道矩阵，256×256×3，其中三个通道分别对应了 R、G、B 的值。

我们要将 RGB 从各 8 位转化为共 12 位的值，我们采用的方法是 RGB 各取其中最高 4 位。将 R 右移 4 位后左移 8 位，G 右移 4 位后左移 4 位，B 右移 4 位。这里右移的作用是舍弃前 4 位以外的数字。再把他们相加得到这一位置的储存值：1100_1001_0101 即 C95

若在 COE 文件中按照原图片的大小格式换行，可以形象地观察。

原图（左）　COE 文件（右）

2. IP 核

我们读入图片的思路实际上是将图片按照一定的顺序，将每个像素点依次存入数组中便于 VGA 逐像素点读取。所以 IP 核实际上就是一个二维数组。

而本次实验要使用的 IP 核就是读入 COE 文件后将每个点存入数组中对应位置的二维数组。并且这个生成的二维数组的调用方式实际上与 Module 较为相似，需要将你要读取的这张图片的坐标传入，然后数组为你返回一个像素值。

```
picture pircture(       //实例化
    .clka(R_clk_25M),   // 本次设备采用的25MHz时钟，我们使用对应时钟
    .addra(R_rom_addr), // input [18 : 0] address地址值
    .douta(W_rom_data)  // output [11 : 0] data   像素值
);
```

我们需要知道，这里的地址实际上是将一张图片先视为二维矩阵，而后拉成一条直线后，像素对应的位置。例如，一张 640×480 的图片，展开后地址会扩展到 307200。

3. VGA 原理

输入参数

vgaclk --------25MHz 时钟信号

d_in --------输出的像素的 RGB12 位颜色值

clrn --------清屏信号，为 0 则本次不输出

输出参数

row_addr ------8 位行坐标（0-479）

col_addr ------9 位列坐标（0-639）

read ------可读信号，1 则输出在显示区域

r，g，b ------3 个 4 位颜色输出

hs ------水平同步信号

vs ------垂直同步信号

VGA 是 IBM 在 1987 年随推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点，彩色显示器领域得到了广泛的应用。

我们大作业使用的屏幕，刷新频率为 60Hz，所以我们需要使用 80052560 = 25.2MHz 的驱动信号，在这里直接使用 25MHz。

在实验中，我们主要需要在一个 VS-HS 完成一个周期的时间内，按从左上角到右下角的顺序输出各个像素点的颜色，就能在显示器上显示出对应的图案。需要注意的是，VGA 颜色的输出必须严格遵循时序，只能在 cd 段输出非 0 数据，否则会对显示造成干扰。VGA 显示依赖于行扫描信号、场扫描信号和当前像素的

RGB 值。电子枪从左上角开始从左向右扫描，

一行扫描完后，进行行消隐（使电子束不再发射，同时将其回归到下一行的最左边，此时行扫描信号为 0），接着扫描下一行，当所有行扫描完后，进行场消隐（电子束回到第一行的最左边，此时场扫描信号为 0），在每次指定的时钟的上升沿，输出场扫描信号、行扫描信号和对应的 RGB 值。

在每一个 vgaclk 的正边沿，水平计数+1，水平计数达到 799 后下一次为 0：

> 当水平计数为 0~95 时，设定同步信号为 0，为同步期，产生行同步脉冲；

> 96~152 为消隐后期，同步信号为 1，但是不能输出数据信号

> 153~792 位显示期，共 640 时钟周期，输出当前行数据信号

> 792~799 为消隐前期，同步信号为 1，不输出数据信号

当水平计数每次到 0 时，说明当前行已经输出完成，垂直计数+1，垂直计数达到 524 后下一次为 0：

> 当垂直计数为 0~1 时，设定同步信号为 0，产生场同步脉冲；

2~42 为消隐后期，同步信号为 1，但是不能输出数据信号

43~522 位显示期，共 480 个，输出当前帧数据信号，在这里输出每一行

523~524 为消隐前期，同步信号为 1，不输出数据信号

这里使用的数据与理想值尽管存在一定的偏差，但是确是我们实验中得到的， 因此我们可以认为这是符合实际的值。

二、实验内容原理

本次游戏设计 top 框架主要包含 8 个模块：

（1）VGA 显示模块

（2）游戏界面

（3）运动模块

（4）跳跃控制

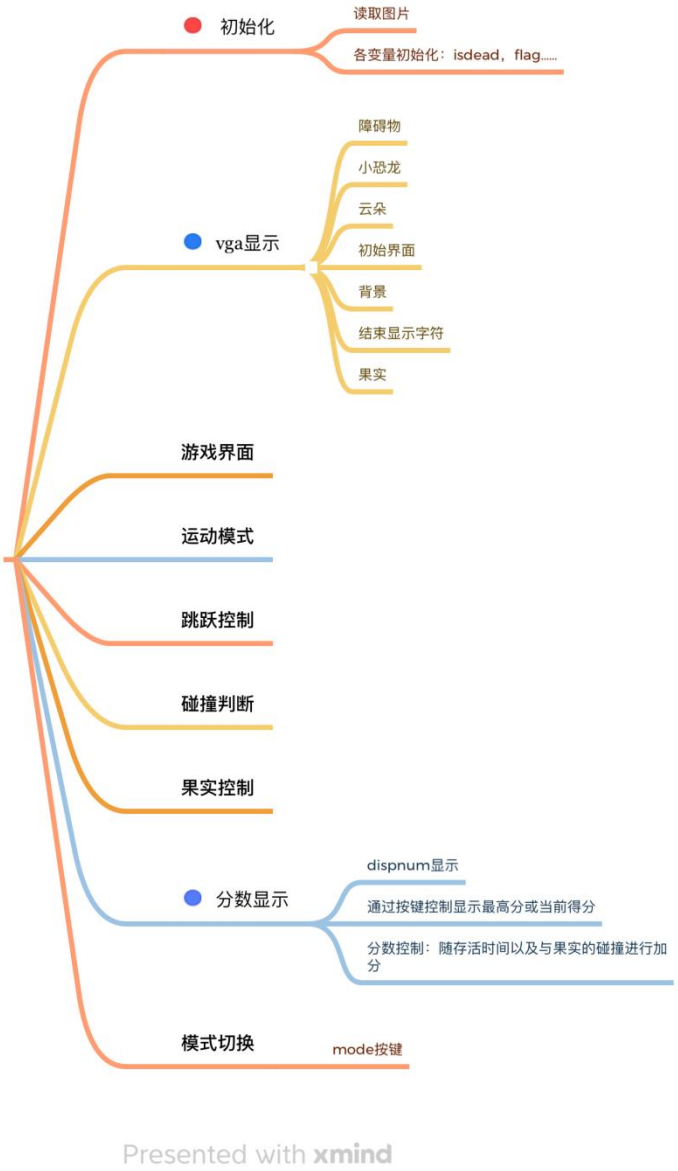
（5）碰撞判断

（6）果实控制

（7）分数显示

（8）模式切换

Top.v



总体原理来说，小恐龙在 x 轴上是固定不运动的，x 轴上运动的只有障碍物、漂浮的云以及加分果实，我们要实现的就是通过控制恐龙 y 轴的变化，也就是跳跃，做到越过障碍物以及吃到加分果实等一系列操作。而游戏的结束依据，也就是小恐龙撞上了障碍物，需要我们用类矩阵的思想，只要小恐龙的矩阵碰撞到了障碍物的矩阵就停止运动。

下面我们将依次介绍主要模块的原理

1. VGA 显示模块

了解了 VGA 运行逻辑后，我们所写的 VGA 模块实际所需的

输入：12 位 RGB 数值，25MHZ 时钟，清除信号

输出：行，列地址、RGB 各 4 位数值，可读信号，水平数值同步信号

PPT 已经给出实现方式，这里强调最关键的代码部分：

```
// vga signals
always @ (posedge vga_clk or negedge clrn)
begin
if (!clrn)
    begin
      row_addr <=  9'b0; // pixel ram row address
    col_addr <=  10'b0;      // pixel ram col address
    rdn      <=  1'b1;     // read pixel (active low)
    hs       <=  1'b0;   // horizontal synchronization
    vs       <=  1'b0;   // vertical   synchronization
    r        <=  4'b0; // 3-bit red
    g        <=  4'b0; // 3-bit green
    b        <=  4'b0; // 2-bit blue
    end
else
begin
    row_addr <=  row[8:0]; // pixel ram row address
    col_addr <=  col[9:0];      // pixel ram col address
    rdn      <= ~read;     // read pixel (active low)
    hs       <=  h_sync;   // horizontal synchronization
    vs       <=  v_sync;   // vertical   synchronization
    r        <=  rdn ? 4'h0 : d_in[11:8]; // 3-bit red
    g        <=  rdn ? 4'h0 : d_in[7:4]; // 3-bit green
    b        <=  rdn ? 4'h0 : d_in[3:0]; // 2-bit blue
end
end
```

这里首先判断了清除信号是否打开，打开则全屏清黑，位置也不再改变。若清除信号并不打开，则为 RGB 分别赋值然后给出已经位移的地址。

2. 游戏界面

我们设计了两个游戏界面，用 state 区分现在界面的状态，当 state = 0 时为开始界面，游戏处于准备状态。当我们点按 start 的按钮后，我们的界面切换到游戏界面，这里面我们将扫描小恐龙、云朵、苹果和障碍物的位置并且按照

Game over 文字>障碍物>小恐龙>云朵>水果>背景图片

的优先级扫描全屏输出像素点。

其中判断条件我们将在运动模块中解释其具体来源。

```
always @(posedge R_clk_25M or negedge I_rst_n)begin
    case(state)
    1'b0:begin
        if(!I_rst_n)
            vga_data<=12'b0;
        else
            vga_data<=data_backg[11:0];
    end
    1'b1:begin
        if(!I_rst_n)
            vga_data<=12'b0;
        else if(col_addr>=320-100&&col_addr<=320+100 &&row_addr>=240-50&&row_addr<=240+50 && Is_dead==1'b1)
            vga_data <= data_gameover;      //gameover文字
        else if(col_addr>=639 - v * t - w&&col_addr<=639 - v * t &&row_addr>=400 - h+ h_dina&&row_addr<=400+ h_dina)
            vga_data<=W_rom_data[11:0];     //障碍物1
        else if(col_addr>=639 - v * t &&col_addr<=639 - v * t +w&&row_addr>=400 - h+ h_dina&&row_addr<=400+ h_dina && flag_of_grass==1'b1)
            vga_data<=W_rom_data[11:0];     //障碍物2
        else if(col_addr>=250 - dinaw&&col_addr<=250 &&row_addr>=322 - dinah + h_dina&&row_addr<=400 - dinah + h_dina)
            vga_data<=data_dina[11:0];      //恐龙
        else if(col_addr>=639 - v * t - w_cl + rand1 &&col_addr<=639 - v * t + rand1 &&row_addr>=50 - h_cl + rand &&row_addr<=50 + rand)
            vga_data<=data_cloud[11:0];     //云
        else if(col_addr>=639 - v*t + fruit_x - w_fruit&&col_addr<=639 - v*t + fruit_x &&row_addr>=250 - h_fruit + fruit_y&&row_addr<=250 + fruit_y && flag_of_fruit1 == 1'b1)
            vga_data<=data_fruit[11:0];     //水果
        else
            vga_data<=data_game;            //背景
    end
    endcase
end
```

而各类 data 类数据都来源于 IP 核，每次时钟上升沿更新一次。

3. 运动模块

实际上我们知道，图片都是一个一个像素打印，所以不存在矩阵，因此我们应用了一个类矩阵的思想。假设存在一个运动的矩阵，当扫描的像素点到达这个矩阵内部则打印我们所需图片形象，这样变向实现了图片运动。这一模块按上述所说，由于不存在真实矩阵，我们只需要在判断条件里面体现“一个运动矩阵”即可。

例如

```
col_addr>=639 - v * t &&col_addr<=639 - v * t +w
```

这里 639 - v t 和 639 - v t + w 描绘了一个以速度 v 已经运动了时间 t 的宽度为 w 的障碍物。

所有物体都有一个逻辑上的矩阵范围，实际上这个范围判定只需要用在判断是否死亡以及 VGA 像素点颜色判断中，其他如变速以及计分模块只需要改变我们的速度 V 和时间 T，我们的矩阵就会随之改变，运动因此实现。

4. 跳跃控制

这里控制跳跃，我们只需要引入一个跳跃按钮 jump_btn，并且提前设置一个恐龙跳跃的高度 dinah 并在类矩阵中计算即可。

在小恐龙没有死亡的前提下判断其是否按下，我们就能给出跳跃状态 flag1 与 flag2，其中 flag1 代表上升过程，每次时钟上升沿我们的跳跃高度 dinah 都会上升 10。当达到了我们设定的高度我们就会接着进入 flag2 状态开始下降，同样每个时钟上升沿都会下降 10，直到下落到地面就会清除我们的状态重新检测跳跃按钮是否被按下。

```
always @(posedge clk_div[20] ) begin
    if(Is_dead==0)begin
        if (jump_btn == 0 && flag2 == 0) begin
            flag1 <= 1;
        end
        if (flag1 == 1 && dinah < 180) begin
            dinah <= dinah + 10;
        end
        else if (flag1 == 1 && dinah >= 180) begin
            dinah <= dinah - 10;
            flag2 <= 1;
            flag1 <= 0;
        end
        else if (flag1 == 0 && flag2 == 1 && dinah > 0) begin
            dinah <= dinah - 10;
        end
        else if (dinah <= 0) begin
            flag2 <= 0;
        end
    end
end
```

5. 碰撞判断

在上面运动模块中，我们知道了每个图片有一个对应的自己对应的类矩阵，在这里，我们将小恐龙的类矩阵与障碍物的类矩阵进行比对，若有重叠部分则说明已经碰撞，我们将更新 Is_dead 状态来暂停游戏，并且显示 gameover 文字（在上面 vga 模块中）。

```
always@(posedge clk_div[1])begin
    if(col_addr>=250 - dinaw&&col_addr<=250 &&row_addr>=322 - dinah + h_dina&&row_addr<=400 - dinah+ h_dina)
        begin
            if(col_addr>=639 - v * t - w &&col_addr<=639 - v * t &&row_addr>=400 - h +h_dina&&row_addr<=400 + h_dina)begin
                gameover1<=1'b1;
                end
            else if(col_addr>=639 - v * t &&col_addr<=639 - v * t +w&&row_addr>=400 - h+ h_dina&&row_addr<=400 + h_dina&& flag_of_grass==1'b1)begin
                gameover1<=1'b1;
                end
            else gameover1<=1'b0;
        end
    else
        gameover1 <= 1'b0;
    if(gameover1 == 1'b0 && state == 1'b0)
        Is_dead <= 1'b0;
    else if(gameover1 == 1'b1 && state == 1'b1)
        Is_dead <=1'b1;
end
```

实际上 gameover 和 Is_dead 可以合成为一个变量，在 state 重新回到开始界面时更新这两个变量。

6. 果实控制

实际上这个部分和障碍物的运动大致一样，只是多了一个果实消失的功能。当小恐龙和果实相碰撞后果实位置重置到界面外，这样就造成了果实的消失。其中 num 将在分数显示中解释。

```
    //fruit get
always @(posedge clk_div[1])begin
    if(col_addr>=639 - v*t + fruit_x - w_fruit&&col_addr<=639 -
        if(col_addr>=250 - dinaw&&col_addr<=250 &&row_addr>=322
            flag_of_fruit1 <= 1'b0;
            flag_of_fruit2 <= 1'b1;
        end
    end
    count <= count + 1'b1;
    if(count == 22'b1111_1111_1111_1111_1111_11)begin
        if(state == 1'b0)num[15:0] <= 16'b0000_0000_0000_0000;
        else if(state == 1'b1 && Is_dead == 1'b0)num <= num + n
        if(flag_of_fruit2 == 1'b1 )begin
            num <= num + 10;
            flag_of_fruit2 <= 1'b0;
        end
        count <= 22'b0;
    end
    if(flag_of_fruit3 == 4)begin
        flag_of_fruit1 <= 1'b1;
        flag_of_fruit2 <= 1'b0;
    end
end
```

7. 分数显示

如上图所示，正常运动时我们使分数 num 每次 clk[22]上升沿时加 1，吃到两种水果时会加 10 分，并且我们用 max_num 记录最大的 num。死亡后 num 停止，点击开始游戏重置 num，但是 max_num 不会重置，始终记录在其中，直到 num 大于它才会继续更新。我们引入一个新的 btn 来切换当前分数和最高分数的显示。要显示的数字我们赋值给 true_num 传给七段管。

```
reg [15:0] num=0;
reg [15:0] max_num = 16'b0000_0000_0000_0000;
reg [15:0] true_num = 16'b0000_0000_0000_0000;
wire [15:0]disp1,disp2;

assign disp1 = num;
assign disp2 = max_num;

always @(posedge clk_div[1])begin
    if(numshow == 1'b0)
        true_num <= disp1;
    else if(numshow == 1'b1)
        true_num <= disp2;
end


always @(posedge clk_div[20])begin
    if(num > max_num)max_num <= num;
end
```

8. 模式切换

为了丰富游戏的可玩性，我们加入了不同的模式。当打开 mode 开关时，小恐龙的速度将会随时间不断增加，直到到达上限，使得游戏的难度不断提高。每次死亡后玩家的速度都将会重新回归正常值。

实验步骤

一、生成 COE 文件

使用 Matlab 写出我们在原理中提到的转化过程，与我给之前给出的 GAMEOVER 图片样例不同的是，实际上的数据储存最好是一个位置的数据占一行，有 n 个数据最终 COE 生成的文件中就有 n+3 行，（n 个数据 n 行，前两行有特殊说明语句，最后一行是空行）例如 640×480 图中有 307203 行。

代码如下：

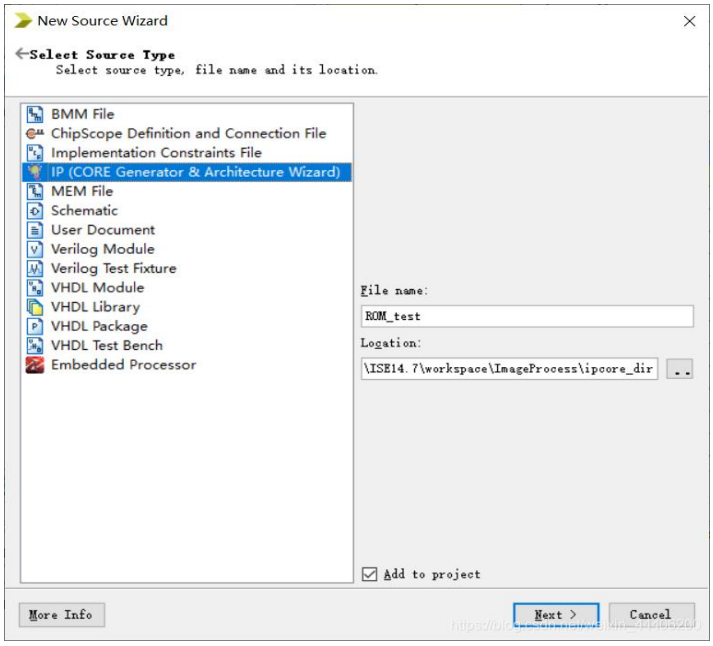
编辑器 - F:\.c\matlab\rgbim24to12.m *

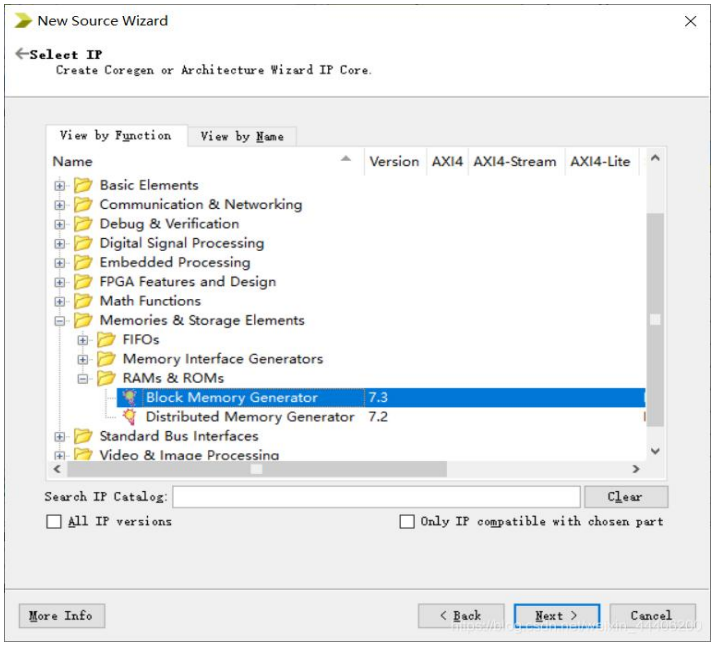
rgbim24to12.m *

```
function rgbim24to12(path,name)
    % 利用imread函数把图片转化为一个三维矩阵
    image_array = imread(path);

    % 利用size函数把图片矩阵的三个维度大小计算出来
    % 第一维为图片的高度，第二维为图片的宽度，第三维为图片的RGB分量
    [height,width,z]=size(image_array);

    red   = image_array(:,:,1); % 提取红色分量，数据类型为uint8
    green = image_array(:,:,2); % 提取绿色分量，数据类型为uint8
    blue  = image_array(:,:,3); % 提取蓝色分量，数据类型为uint8

    % 把上面得到了各个分量重组成一个1维矩阵，由于reshape函数重组矩阵的
    % 时候是按照列进行重组的，所以重组前需要先把各个分量矩阵进行转置以后再重组
    % 利用reshape重组完毕以后，由于后面需要对数据拼接，所以为了避免溢出
    % 这里把uint8类型的数据扩大为uint32类型
    r = uint32(reshape(red'   , 1 ,height*width));
    g = uint32(reshape(green' , 1 ,height*width));
    b = uint32(reshape(blue'  , 1 ,height*width));

    % 初始化要写入.coe文件中的RGB颜色矩阵
    rgb=zeros(1,height*width);

    % 因为导入的图片是24-bit真彩色图片，每个像素占用24-bit，其中RGB分别占用8-bit
    % 而我这里需要的是12-bit，其中R为4-bit，G为4-bit，B为4-bit，所以需要在这里对24-bit的数据进行重组与拼接
    % bitshift()函数的作用是对数据进行移位操作，其中第一个参数是要进行移位的数据，第二个参数为负数表示向右移，为
    % 正数表示向左移，更详细的用法直接在Matlab命令窗口输入 doc bitshift 进行查看
    % 所以这里对红色分量先右移4位取出高4位，然后左移11位作为ROM中RGB数据的第11-bit到第8-bit
    % 对绿色分量先右移4位取出高4位，然后左移5位作为ROM中RGB数据的第7-bit到第4-bit
    % 对蓝色分量先右移4位取出高4位，然后左移0位作为ROM中RGB数据的第3-bit到第0-bit
    for i = 1:height*width
        rgb(i) = bitshift(bitshift(r(i),-4),8) + bitshift(bitshift(g(i),-4),4) + bitshift(bitshift(b(i),-4),0);
    end

    fid = fopen( name , 'w+' );

    % .coe文件的最前面一行必须为这个字符串，其中16表示16进制
    fprintf( fid, 'MEMORY_INITIALIZATION_RADIX=16;\n');

    % .coe文件的第二行必须为这个字符串
    fprintf( fid, 'MEMORY_INITIALIZATION_VECTOR=\n');

    for i = 1:height*width - 1
        fprintf( fid, '%x,\n',rgb(i));
    end
    fprintf( fid,'%x;\n',rgb(height * width));

    fclose( fid ); % 关闭文件指针
end
```

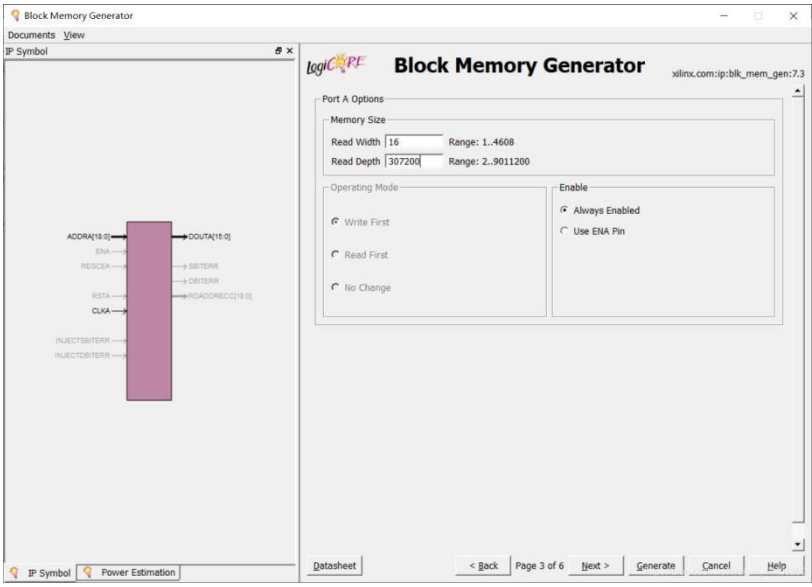
二、生成 IP 核

在 ISE 工程中，点击右键新建文件，选择 IP 核生成选择 Memories 选项中的 RAMs&ROMs 里的 Block Memory Generator

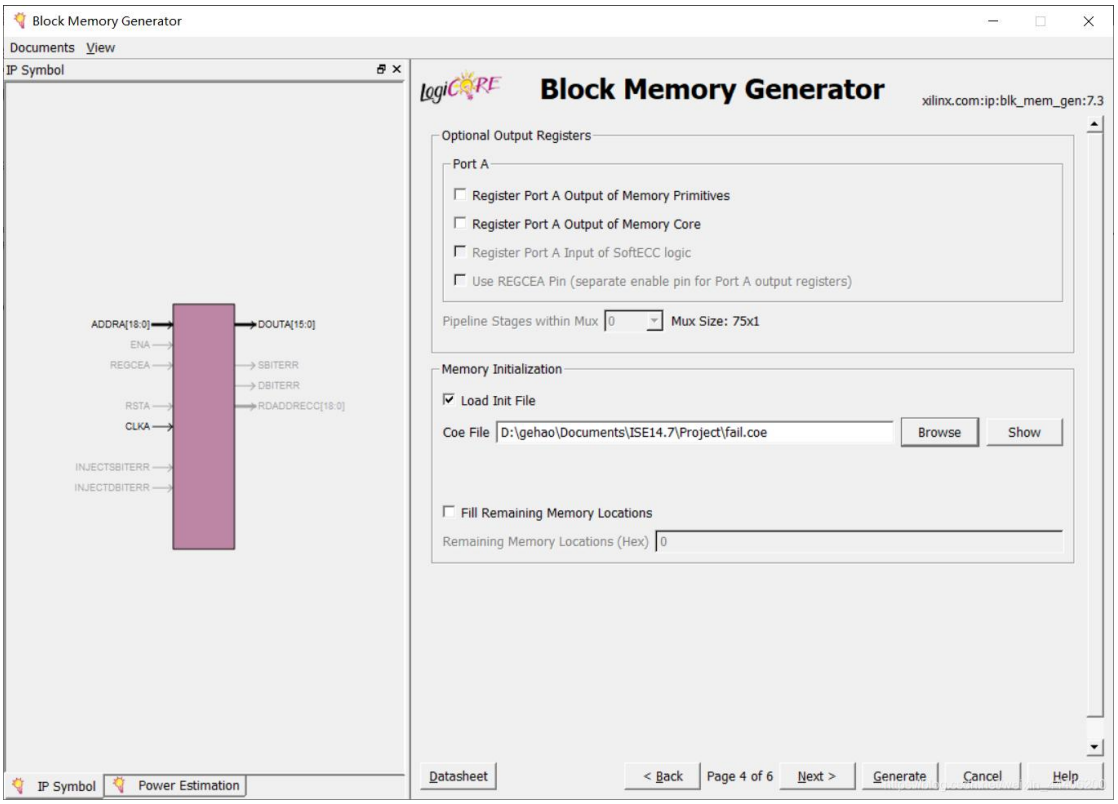




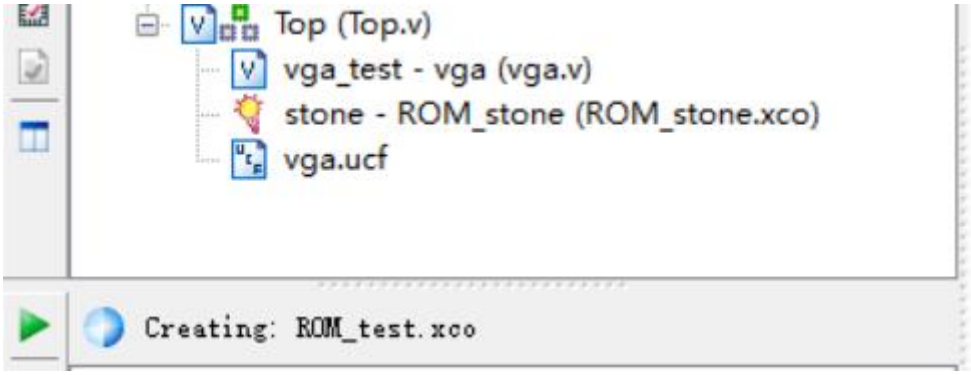
接下来会进入这个界面，一路点击 next。然后进入到这个设置参数的界面其中 width 代表你设置的图片数据是几进制，由于之前写 COE 的时候是用 16 进制写的，因此这里写 16；depth 表示图像的大小，即长乘宽，由于我们使用的是 640480 的图片，因此这里填 307200(=640480)



然后进入这个页面，点击 load init file，选择你的 coe 文件即可；（耐心等待，时间根据你导入图片大小不同而不同，实测机房电脑 640×480 图片生成核需要 30min 左右）



生成完毕的标志是：Creating 结束，并且在你的目录下出现类似于太阳的这个标记



调用在原理中已经说明，类似于 module。

三、 top 编写

首先确定输入与输出

输入：	时钟信号		I_clk
	初始化信号		I_rst_n
	数字显示切换信号		numshow
	游戏开始按钮		start_game
	跳跃按钮		jump
	模式切换		mode
输出：	RGB 三种颜色信号	[3:0]	O_red
		[3:0]	O_green
		[3:0]	O_blue
	水平与竖直扫描信号		O_hs
			O_vs
	七段管输出相关	[7:0]	Segment
		[3:0]	AN
	按钮行输出		BTNX4

接着初始化我们的主要变量，尤其是将两个按钮进行防抖动操作。

```
assign BTNX4 = 0;
pbdebounce m1000(clk_div[16],start_game,start_pb);
pbdebounce m1010(clk_div[16],jump,jump_btn);
```

然后根据我们写好的游戏原理填入相对应的代码。

其他相关模块：clk_div，pbdebounce，dipnum，vga。

接着引入我们已经转换好的 IP 核备用，包括了障碍物、小恐龙、游戏背景、开始游戏界面、水果和云。

```
  barrier barrier(
.clka(R_clk_25M), // input clk_25MHZ
.addra(R_rom_addr), // input [18 : 0] address
.douta(W_rom_data) // output [11 : 0] data
  );

  start start(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_backg), // input [18 : 0] address
.douta(data_backg) // output [11 : 0] data
  );

  dino dino(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_dina), // input [18 : 0] address
.douta(data_dina) // output [11 : 0] data
  );

  cloud cloud(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_cloud), // input [18 : 0] address
.douta(data_cloud) // output [11 : 0] data
  );
```

```
  cloud cloud(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_cloud), // input [18 : 0] address
.douta(data_cloud) // output [11 : 0] data
  );


  fruit fruit(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_fruit), // input [18 : 0] address
.douta(data_fruit) // output [11 : 0] data
  );

  playground playground(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_game), // input [18 : 0] address
.douta(data_game) // output [11 : 0] data
  );
  gameover0 gameover0(
.clka(R_clk_25M), // input clk_25MHZ
.addra(addr_gameover), // input [18 : 0] address
.douta(data_gameover) // output [11 : 0] data
  );
```

引脚接入

```
NET "O_blue[0]" LOC = T20 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_blue[1]" LOC = R20 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_blue[2]" LOC = T22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_blue[3]" LOC = T23 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_green[0]" LOC = R22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_green[1]" LOC = R23 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_green[2]" LOC = T24 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_green[3]" LOC = T25 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_red[0]" LOC = N21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_red[1]" LOC = N22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_red[2]" LOC = R21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_red[3]" LOC = P21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_hs" LOC = M22 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "O_vs" LOC = M21 | IOSTANDARD = LVCMOS33 | SLEW = FAST ;
NET "I_clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET "I_rst_n" LOC = AA10 | IOSTANDARD = LVCMOS18;
NET "jump" LOC = V14 | IOSTANDARD = LVCMOS18;
NET "numshow" LOC = AA13 | IOSTANDARD = LVCMOS18;
NET "mode" LOC = AA12 | IOSTANDARD = LVCMOS18;
NET "jump" CLOCK_DEDICATED_ROUTE = FALSE;
NET "start_game" LOC = W14 | IOSTANDARD = LVCMOS18;
NET "start_game" CLOCK_DEDICATED_ROUTE = FALSE;
NET "BTNX4" LOC = W16 | IOSTANDARD = LVCMOS18;

NET "Segment[0]" LOC = AB22 | IOSTANDARD = LVCMOS33;
NET "Segment[1]" LOC = AD24 | IOSTANDARD = LVCMOS33;
NET "Segment[2]" LOC = AD23 | IOSTANDARD = LVCMOS33;
NET "Segment[3]" LOC = Y21 | IOSTANDARD = LVCMOS33;
NET "Segment[4]" LOC = W20 | IOSTANDARD = LVCMOS33;
NET "Segment[5]" LOC = AC24 | IOSTANDARD = LVCMOS33;
NET "Segment[6]" LOC = AC23 | IOSTANDARD = LVCMOS33;
NET "Segment[7]" LOC = AA22 | IOSTANDARD = LVCMOS33;

NET "AN[0]" LOC = AD21 | IOSTANDARD = LVCMOS33;
NET "AN[1]" LOC = AC21 | IOSTANDARD = LVCMOS33;
NET "AN[2]" LOC = AB21 | IOSTANDARD = LVCMOS33;
NET "AN[3]" LOC = AC22 | IOSTANDARD = LVCMOS33;
```

实验结果

在本项目中，我们成功地整合了各个先前实现的模块，实现了一个富有趣味性的小恐龙跑酷游戏。游戏具备多项功能，其中包括小恐龙的连续奔跑、跳跃按钮按下时的跳跃效果，以及在撞到障碍物时的死亡效果。通过按键控制，玩家可以轻松地掌控小恐龙的行动，通过跳跃按钮（jump）实现起跳，而在游戏结束后，通过按下重新开始按钮（rst）重新开始游戏。

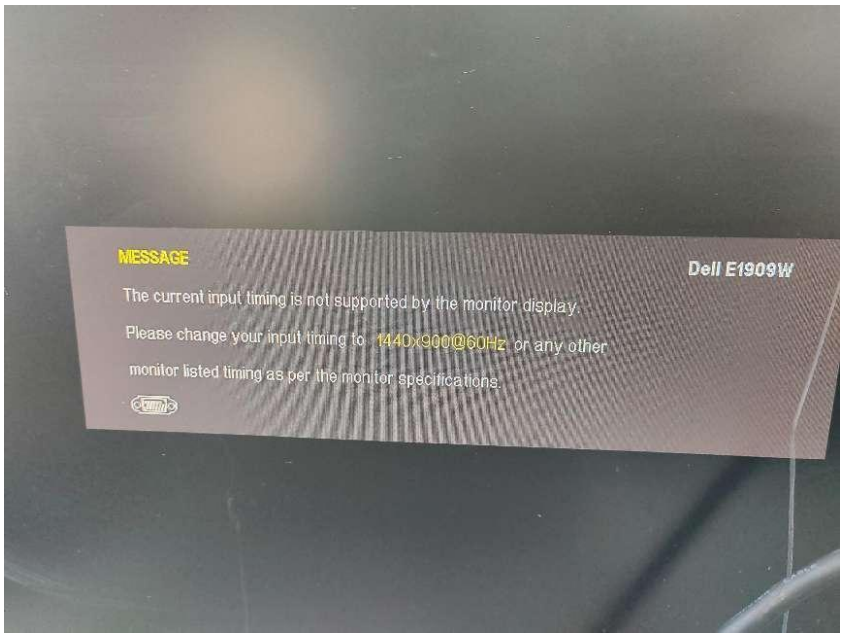
实验中还引入了模式选择的功能，通过开关控制实现了游戏的不同模式。这样的设计增加了游戏的可玩性，使玩家能够在不同的挑战模式中体验游戏的乐趣。此外，VGA 游戏界面以及实时显示当前得分或最高得分，为玩家提供了清晰而直观的游戏信息。

在演示视频中，我们展示了游戏的各项功能，并突出了按键操作的流畅性以及游戏画面的精美设计。通过本次实验，我们成功地将各个模块有机地组合在一起，实现了一个富有娱乐性和挑战性的小恐龙跑酷游戏，为游戏开发和嵌入式系统设计提供了有益的经验和教训。

Debug 经验总结

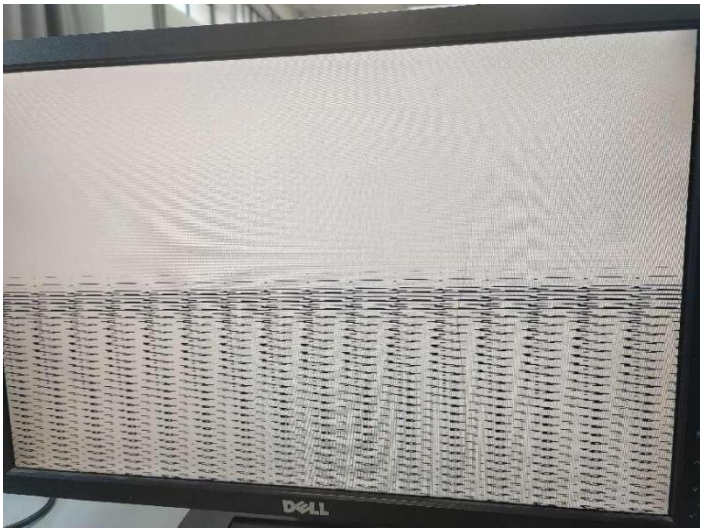
一、时钟不匹配

本次实验使用设备为 25MHZ 频率，而对应原来系统自带时钟为 50MHZ，我们使用 clk_div 后调用 clk[1]即可。



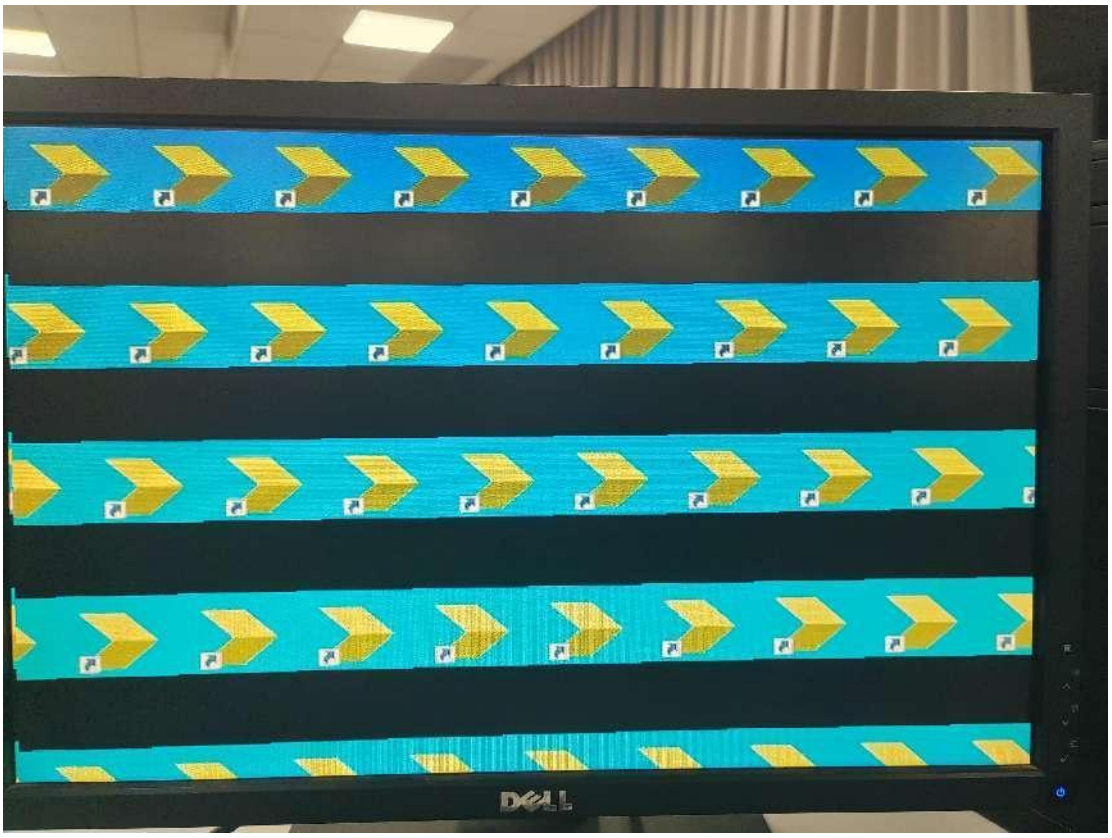
二、屏幕下半部分集中图片乱码

所用的图片地址和屏幕地址没有对齐，需要仔细查看后修改。



三、图片多重显示

没有框定 vag_data 被图片赋值的范围，使得本不该出现的图片显现出来。



四、运动时物体逐层扫描的视觉问题

此问题没有拍照，但实际上运动时呈现出物体有一个从上到下循环扫描的样子，这个问题在不使用像素赋值而用 IP 核导入的图片赋值后得到解决。（原因不明，推测可能是扫描频率与移动频率出现重合导致）

实验总结以及心得体会

通过本次实验，我们对数字逻辑设计有了更深的理解，对于 Verilog 语言的使用也更加熟练。同时，我们也对于数字逻辑设计的实际应用有了更深的认识。

首先是本次实验完成过程的不足之处，由于准备时间较为仓促且具体的资料也较少，我们部分细节之处的处理不够完美，同时也有一些功能没有实现，比如说我们的小恐龙没有实现**下蹲的功能**，同时也没有实现小恐龙**吃到果实的动画效果**，以及可以实现**更好的分数显示**还有**障碍物的上下分级**。

其次是我们在实验过程中的收获，本次工程的参考较少，我们相对独立的完成了项目的构思以及各个环节之中的变量处理以及内存的分配，这些都与我们在课堂上所学习到的知识息息相关，通过这次实验，我们对于这些知识有了更深的理解，也对于前面的实验有了更深的认识。

最后，总的来说我们的实验完成度还是较高的，我们的实验完成了基本的功能，同时也有一些额外的功能，比如说我们的游戏可以实现两种模式的切换，同时也可以实现小恐龙吃到果实的额外得分，这些都是我们自己添加的功能，这些功能的实现也是我们对于数字逻辑设计的理解的体现。

我们相信，在未来的学习过程中，我们一定能发挥的更好，学到更多的知识，在计算机组成中更好地面对接下来的挑战。

工作量分布：

徐梓毓：编写运动、随机、碰撞、果实、分数的相关代码以及实验背景与总结以及部分原理的论文以及上板。

杨辰：　编写 VGA 图片显示、游戏界面、分数显示、COE 文件与 IP 核相关以及剩余实验报告部分。