Digital Logic Project

Flappy Bird Design Report

Tan Qiye 3150105197

Date: 2018-1-7

Section 1：简介

1.1 主要功能

这个项目主要是在 sword 板上复现经典游戏：flappy bird。在这个游戏中，你可以通过 sword 板自带按钮来控制小鸟飞行，需要注意的是，你只能控制小鸟向上飞行，小鸟的下降是由重力决定的，按钮没有被按下时，小鸟会以重力加速的模型下降。你要控制小鸟一直向前飞行，然后注意躲避途中高低不平的管子。当小鸟撞上柱子、飞得过高（撞上屏幕上边沿）或是掉落到地面时，游戏结束。每当你成功穿越一个柱子，你的分数会加一并显示在 7 段数码管上。

1.2 为什么会选择这个项目

《flappy bird》是一款由来自越南的独立游戏开发者 Dong Nguyen 所开发的作品，玩法简单，但是在短短几天之内几乎占据了 80%的欧美手机游戏用户。因此，网络上存在大量该游戏相关素材，而且个人感觉复现这样一款游戏挺酷的，同时实现它的难度也不算太高。

实际上，在游戏逻辑上，flappy bird 只需要实现物体坐标的移动以及碰撞检测即可。为了增加一点难度，我增加了小鸟扑腾翅膀的动态效果，让这个 project 更接近原生 flappy bird 的效果。虽然理论上逻辑并不复杂，但是使用 Verilog 实现并进行像素级别的操作还是挺繁琐的，幸运的是我最终成功地解决了所有 bug 并成功复现了整个游戏。

Section 2：设计规格

2.1 开发环境

开发平台
  Family: Kintex7
  Device: XC7K160T
  Package: FFG676
开发软件
  Xilinx ISE 14.7
开发语言
  Verilog

2.2 运行要求

这个游戏需要两部分硬件来运行，一个是 2.1 所示的开发板，一个是 VGA 接口显示屏。

Input

SW [15:0]: 开发板上的 16 个 switch

BTN_X [4:0], BTN_Y [3:0]: 开发板右侧的按钮矩阵

Output

显示屏：显示游戏界面

7 段数码管：显示游戏运行时玩家所得分数

Section 3: 模型设计

3.1 设计模式

在这个游戏中，屏幕上需要显示小鸟、柱子、地面以及背景。为了解决这些数据和显示相关逻辑，我运用了一点 MVVM 的设计模式来设计游戏的逻辑架构。MVVM 设计模式主要有三个部分：Model，View 以及 View Model。

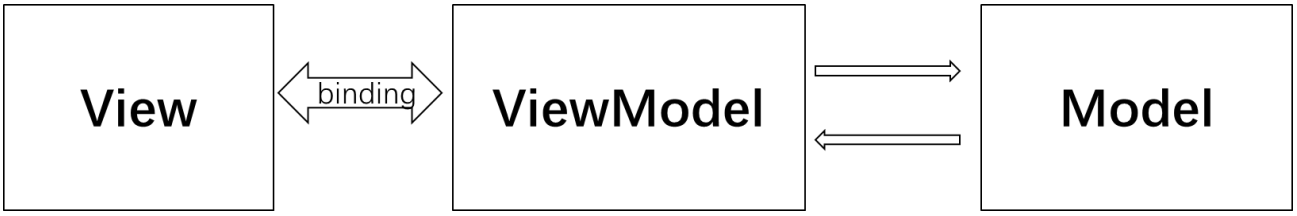


Figure 1 MVVM structure

Model 指的是代表真实状态内容的领域模型（面向对象方法），或代表内容的数据访问层（以数据为中心的方法）。 在我的游戏中，各物体运动的主要逻辑就在这个部分。

View 是用户在屏幕上看到的结构，布局和外观。VGA 模块可以从 View Model 获取数据并在显示器上显示外观。

View model 是暴露公共属性和命令的视图的抽象。MVVM 在 View 和 View Model 之间有一个绑定。在 View Model 中，这种绑定介导了 View 和数据之间的通信。View Model 被描述为模型中数据的状态。这部分涉及显示逻辑。

3.2 Top module

这部分是整个游戏的顶层设计部分，在这里实现了 View Model，将下层各模块的数据通过显示逻辑绑定到 View 上。此外，该模块还涉及游戏运行开关以及游戏运行状态相关的逻辑。

Input

clk,

SW [15:0]

rstn

BTN_X [4:0]

BTN_Y [3:0]

Output

hs,
vs,
r [3:0]
g [3:0]
b [3:0]
buzzer
AN[3:0]
SEGMENT[7:0]

Function

控制游戏正常运行

整个游戏运行流程如下：

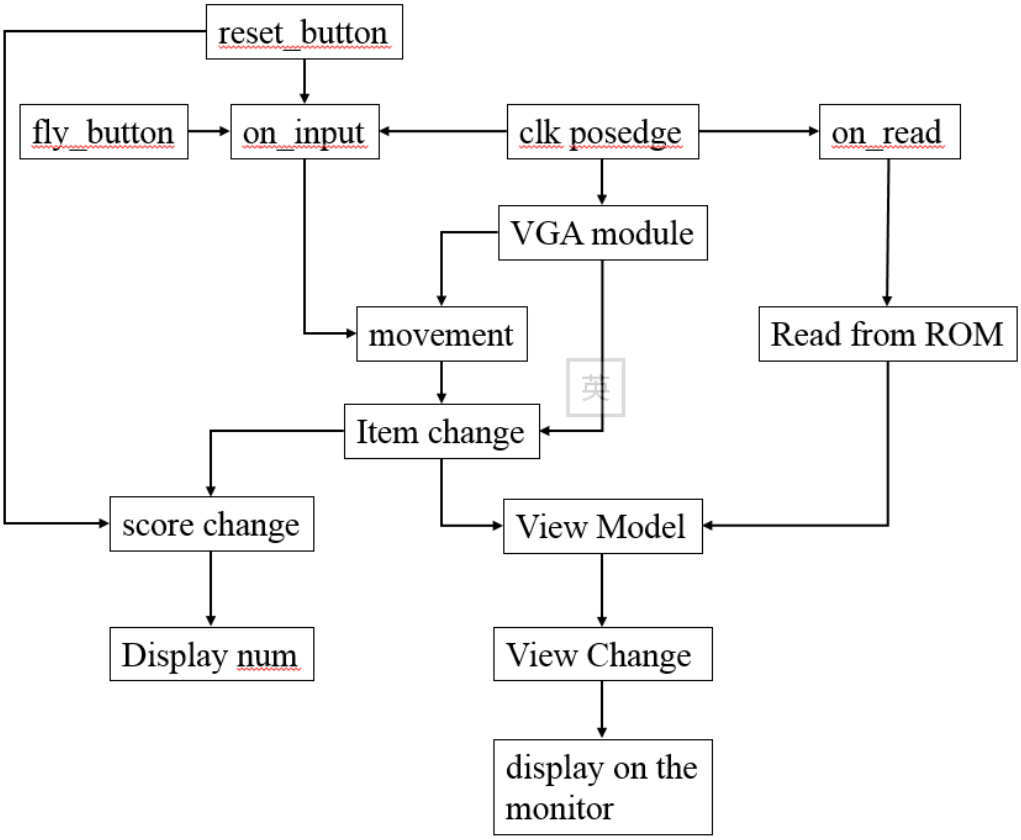


Figure 2 data flow

如上图所示，每一个时间点，时钟将触发读取，输入状态并介导 VGA 模块扫描信号。VGA 模块接受时钟信号后能产生行同步和列同步信号并输出屏幕坐标 x 和 y 给各个逻辑模块。如果 button 被按下，该变化会被触发传入各个 item 逻辑模块（也就是 movement）并导致 item 坐标改变。同时，没有按钮按下时，小鸟要向前飞行且以重力加速度模型降落，时钟还将通过 VGA 模块介导一部分 item 逻辑模块随时间的坐标变化。View Model 从各 item 模块取出描述性数据和从 ROM 取出像素点 RGBA 数据进行逻辑处理，用多路选择器实现图层覆盖，然后呈现出 View，并交给 VGA 模块在屏幕上展示出来。

游戏运行时的得分数据由柱子模块介导，当柱子被小鸟穿越过后，分数增加。此外，reset

按钮可以将分数清零。

除此之外，游戏开关以及碰撞检测都在这个模块实现。碰撞检测是通过接受各模块坐标数据进行逻辑分析得到的。

显示部分伪代码：

```
assign r = vga_vedio_on ? 4'h0 :
           (is_bird && transparency != 0) ? bird_color[3:0] :
           (is_ground && transparency != 0) ? ground_color[3:0] :
           (is_column_up && transparency != 0) ? pipe_up_color[3:0] :
           (is_column_down && transparency != 0) ? pipe_down_color[3:0] :
           bk_color[3:0];

assign g = vga_vedio_on ? 4'h0 :
           (is_bird && transparency != 0) ? bird_color[7:4] :
           (is_ground && transparency != 0) ? ground_color[7:4] :
           (is_column_up && transparency != 0) ? pipe_up_color[7:4] :
           (is_column_down && transparency != 0) ? pipe_down_color[7:4] :
           bk_color[3:0];

assign b = vga_vedio_on ? 4'h0 :
           (is_bird && transparency != 0) ? bird_color[11:8] :
           (is_ground && transparency != 0) ? ground_color[11:8] :
           (is_column_up && transparency != 0) ? pipe_up_color[11:8] :
           (is_column_down && transparency != 0) ? pipe_down_color[11:8] :
           bk_color[3:0];
```

碰撞检测伪代码：

```
assign trigger_stop = ((is_bird && transparency != 0) && is_column_down) ||
                      ((is_bird && transparency != 0) && is_column_up) ||
                      ((is_bird && transparency != 0) && is_ground) ||
                      (is_bird && is_top);
```

计算分数伪代码：

```
always@(posedge RESET or negedge score_out)begin
    if(RESET) begin
        score = 0;
    end else begin
        score++;
    end
end
```

3.3 VGA moule

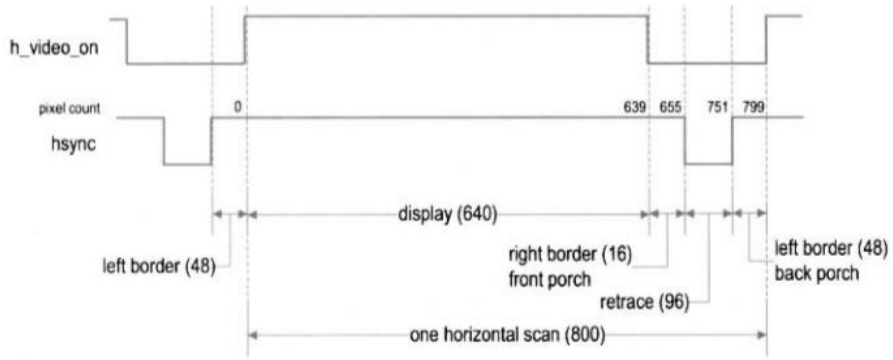
Input

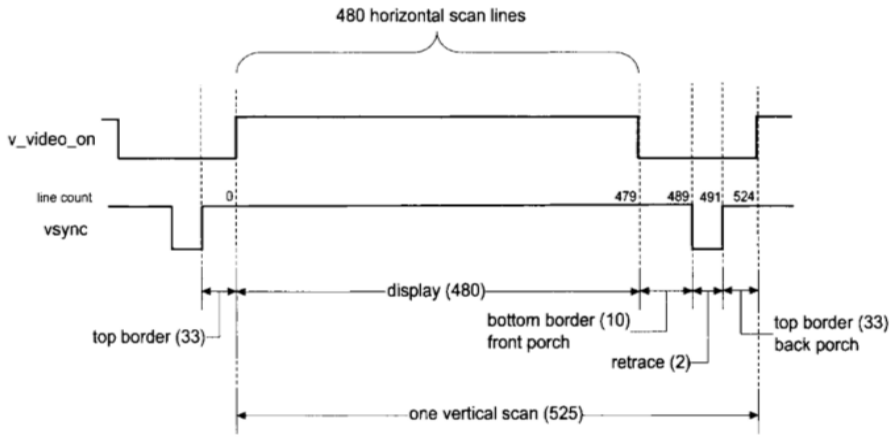
vga_clk: 25MHz
clrn: 使能信号

Output

rdn：当 rdn = 1 时为显示状态，rdn = 0 时为消隐期
row_addr [8:0]：行地址，等价于纵坐标。
col_addr [9:0]：列地址，等价于横坐标。
hs：行同步信号
vs：列同步信号

Function





以上是水平扫描和垂直扫描的时序图。 当一个 clk 信号到来时，把 h_count 加一，如果一行已经完成，将 v_count 加 1，并将 h_count 复位为 0。因此，可以逐行扫描显示器来显示图像。同时要检查同步信号。如果 h_count 和 v_count 在显示行中，则应将同步信号设置为 1。如果 h_count 和 v_count 在消隐期中，则将 rdn 设置为 1。

以下是伪代码：

```
module Vga(
    input clrn;
    input vga_clk;    // 25MHz
    output reg [8:0] row_addr; // pixel ram row address, 480 (512) lines
    output reg [9:0] col_addr; // pixel ram col address, 640 (1024) pixels
    output reg rdn;           // vga_vedio_on
    output reg hs,vs;        // horizontal and vertical synchronization
);

    reg [9:0] h_count;
    reg [9:0] v_count;

    always @(posedge clk)
    begin
        if (!clrn) begin
            set h_count to be 0;
        end else if (h_count reaches the end)
        begin
            set h_count to be 0;
        end
        else
        begin
            h_count++;
        end
    end

    always @(posedge clk or negedge clrn)
    begin
        if (!clrn) begin
            set h_count to be 0;
        end else if (h_count has reached the end)
        begin
            if (v_count reaches the end)
            begin
                set v_count to be 0;
            end
            else
            begin
                v_count++;
            end
        end
    end

    [9:0] row    =    v_count - 35;
```

```
    [9:0] col   =   h_count - 143;
    h_sync = (h_count > 95);
    v_sync = (v_count > 1);
    read = (h_count > 142) &&
           (h_count < 783) &&
           (v_count > 34)   &&
           (v_count < 515);
endmodule
```

3.4 Bird module

Input

fresh：每一帧
x [9:0]：横坐标
y [8:0]：纵坐标
fly_button：飞行按钮
RESET：重启开关
START：开始按钮
Lose：结束状态
game_status：游戏状态
clkdiv [31:0]：32 位分频时钟

Output

is_bird：当前坐标是否应该显示小鸟图片
bird_y [8:0]：小鸟高度

Function

这部分主要控制小鸟的运动模块。当游戏处于运行状态时，小鸟一直处于重力场中，fly_button 没有被按下时要下降，当 fly_button 被按下后，速度瞬间变为向上。此外，小鸟的横坐标保持不变，向前飞行的效果由柱子和地面模块的相对运动来实现。

以下是伪代码：

```
module Bird(
    input wire fresh,
    input wire [9:0] x,
    input wire [8:0] y,
    input wire fly_button,
    input wire RESET,
    input wire START,
    input wire Lose,
    input wire game_status,
    input wire [31:0] clkdiv,
    output reg is_bird,
    output reg [8:0] bird_y
```

```
);

parameter bird_x = 120;
parameter a = 1; // the acceleration of gravity

always @( negedge fresh) begin
      if (game_status) begin
          if (fly_button) begin              //if the fly button is pressed
              velocity = -4;
              bird_y <= bird_y + velocity;
          end
          else begin
              velocity = velocity + a;
              bird_y = bird_y + velocity;
          end
      end else begin
          if ( (~Lose) || RESET) begin  //when begin or reset
              bird_y = 240;
              velocity = 0;
          end
      end
end

always @( posedge clkdiv[0]) begin
      if (x, y belongs to bird section) begin
          is_bird = 1;
      end else begin
           is_bird = 0;
      end
end
```

3.5 Ground module

Input

clkdiv [31:0]：32 位分频时钟
fresh：每一帧
x [9:0]：横坐标
y [8:0]：纵坐标
game_status：游戏运行状态

Output

is_ground：判断当前坐标是否属于地面部分

speed [3:0]：移动速度
ground_position [9:0]：移动后的地面坐标

Function

接受坐标参数和运行状态参数，分析地面的运动状态，并输出运动后的坐标以及地面的移动速度。因为地面是由 336px * 112px 图片拼接而成，所以每次移动的 x 坐标要对 336 求余。

以下是伪代码：

```
module Bird(
    input wire [31:0]clkdiv,
    input wire fresh,
    input wire [9:0] x,
    input wire [8:0] y,
    input wire game_status,
    output reg [9:0] ground_position,
    output reg [3:0] speed,
    output reg is_ground
);

always @( negedge fresh) begin
    if (game_status) begin
        ground_position = (ground_position + speed)%336;
    end
end

always @( posedge clkdiv[0]) begin
    if (x, y belongs to ground section) begin
        is_ground = 1;
    end else begin
        is_ground = 0;
    end

    if (game_status == 0) begin
        speed = 4;
    end
end
```

3.6 Column module

Input

clkdiv [31:0]：32 位分频时钟
fresh：每一帧
x [9:0]：横坐标
y [8:0]：纵坐标

game_status：游戏运行状态
speed [3:0]：移动速度
RESET：重启开关
START：开始按钮

Output

score_out：分数计数触发器
is_column_up：当前坐标是否属于朝上柱子部分
is_column_down：当前坐标是否属于朝下柱子部分
pipe_x [9:0]：柱子横坐标
pipe_y [8:0]：柱子纵坐标

Function

实际上，每一帧的画面上都有两个柱子，因此模块内部要计算好当前帧各个柱子的坐标，以及当前 pixel 是否属于柱子部分，属于哪个柱子部分

以下是伪代码：

```
module Bird(
    input wire fresh,
    input wire [31:0] clkdiv,
    input wire [9:0] x,
    input wire [8:0] y,
    input wire RESET,
    input wire START,
    input wire game_status,
    input wire [3:0] speed,
    output reg score_out,
    output reg is_column_up,
    output reg is_column_down,
    output reg [9:0] pipe_x,
    output reg [8:0] pipe_y
);

//four pipes
reg [9:0] pipe_x1 = 10'd650;
reg [9:0] pipe_x2 = 10'd970;
reg [9:0] pipe_y1=10'd315;
reg [9:0] pipe_y2=10'd145;

always @( negedge fresh) begin
    if (game_status) begin
        pipe_x1 = pipe_x1 - speed;
        pipe_x2 = pipe_x2 - speed;
```

```
            if (pipe_1 move out of the screen) begin
                set the height of next pipe
            end

            if (pipe_2 move out of the screen) begin
                set the height of next pipe
            end

            if (the bird cross the left of pipe) begin
                score_out = 1;
            end
            else if (the bird cross the right of pipe) begin
                score_out = 0;
            end
        end else begin
            if ( (~Lose) || RESET) begin  //when begin or reset
                reset the pipe_x1, pipe_x2, pipe_y1, pipe_y2 and score_out
            end
        end
end

always @( posedge clkdiv[0]) begin
        if (x, y belongs to up pipe 1) begin
            set is_column_down, is_column_up, pipe_x, pipe_y;
        end else if (x, y belongs to down pipe 1) begin
            set is_column_down, is_column_up, pipe_x, pipe_y;
        end else if (x, y belongs to up pipe 2) begin
            set is_column_down, is_column_up, pipe_x, pipe_y;
        end else if (x, y belongs to down pipe 2) begin
            set is_column_down, is_column_up, pipe_x, pipe_y;
        end
end
```

Section 4: 仿真

4.1 Top module & item module

顶层和各模块是实现游戏主要逻辑的地方，包括坐标计算，图层显示，分数计算，碰撞检测，运行状态管理等等。不得不承认，这是非常复杂的，我没有撰写激励代码去对整个顶层模块进行仿真。相反，我通过将其下载到电路板并操作游戏来调试该部分。

4.2 VGA module

以下是激励代码：

```
initial begin
    clrn = 1;
end

always begin
    vga_clk = 0;#1;
    vga_clk = 1;#1;
end
```

激励波形

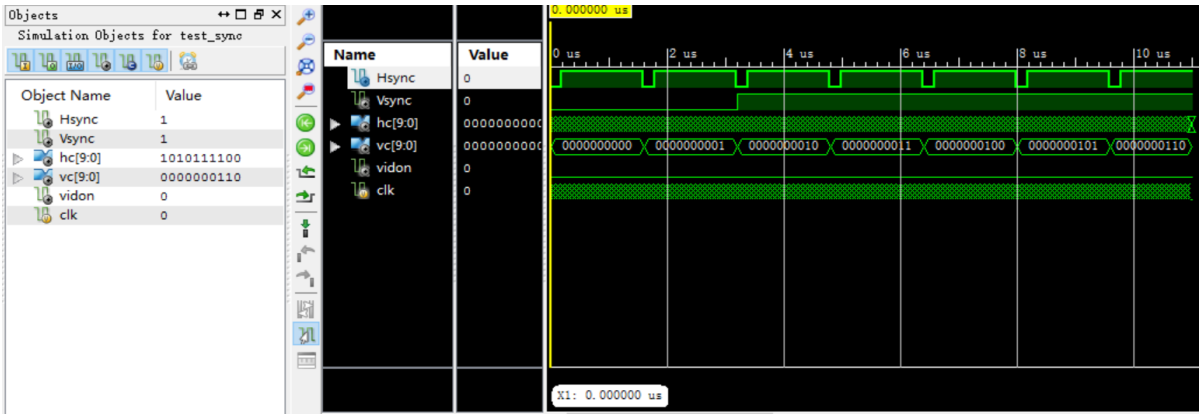


Figure 3 Simulation of vga_sync (1)

我们可以在波形中看到 hs 从 1 变为 0，与 Section 3 中的时序图相同。vs 在开始时变为 1，在整个监视器被覆盖之前不会变为 0。h_count 变化比其他变化更快（与 clk 同步）。我们可以看到每次 v_count 加 1，在这里我们看不到 rdn 是 1，因为刚开始处于边界上，属于消隐期。如果我们让激励代码运行更长的时间，我们可以在 v_count > 34 之后看到 rdn = 1，如下图所示。

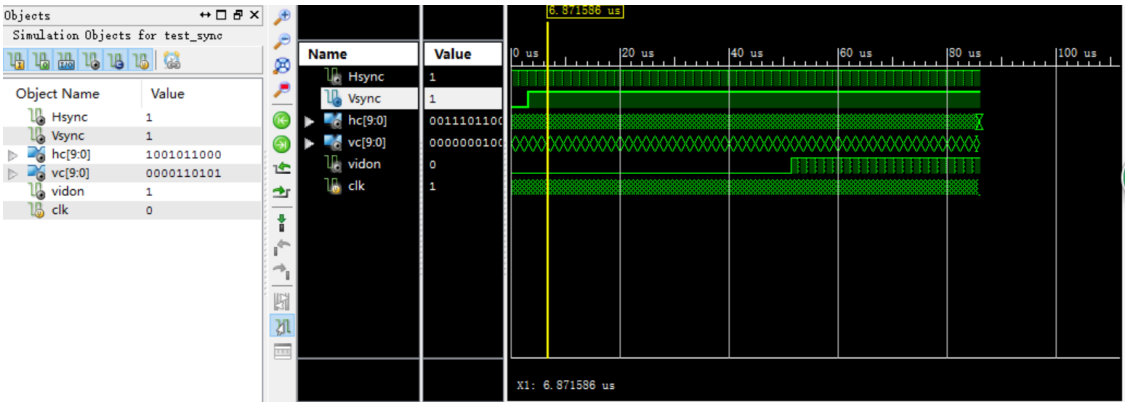


Figure 4 Simulation of vga_sync (2)

Section 5：调试过程分析

5.1 Top module & items module

顶层模块主要涉及 View Model 部分，学会用 assign 做多路选择器后，整体代码虽然多，但并不复杂，基本一次即通过。

Ground 模块相比而言也不复杂，主要计算一下随时间运动的偏移量即可，一次通过。

Bird 模块逻辑稍微复杂一些，需要考虑重力加速度等逻辑。但是，靠地面和柱子相对向后移动造成小鸟向前移动的效果大大降低了小鸟的逻辑复杂度。因此，在这个模块，我可以将小鸟的横坐标固定，只要处理小鸟纵坐标的逻辑即可。在模块内设置速度变量和加速度变量，在每一帧到来时都让速度加一次加速度，再让纵坐标加一次速度，当按下 fly_button 时让速度瞬间变为正的，这样即实现了重力加速度模型。想清楚了以后，整份代码一气呵成，一次性通过。

最难的部分是 Column 模块，因为在一个页面中要显示两个柱子，因此处理起来就会有些繁琐。

Column 第一个难点在于，一开始，我不清楚 Verilog 本身自带补码表示负数的系统，因此很多地方要靠求余等操作计算，导致逻辑混乱，最后跑出来的结果是两个柱子撞在了一起，柱子脱离了原本设想的位置。后来通过查资料得知 Verilog 自带补码系统，可以直接将两个数进行加减，这样一来，前面的问题就都不存在了。

Column 模块第二个难点在于如何确定柱子的高度，Verilog 是严格的硬件语言，很难去设置一个随机数或类似的东西让柱子的高度不断变化。因此我最后的设计是让柱子的高度不断增加一定数值然后对一个数值求余，这就做了一个伪随机的感觉。这之后，又出现了一个问题就是，柱子的素材图片高度只有 320px，因此，如果柱子的高度太高，底端就无法显示出来了。因此，我又做了弥补，每当柱子高度过高时，就重新给高度赋值。

综上，使用了比较好的设计模式之后，可以让业务逻辑更加清晰，仔细思考后写代码，基本上不会出现大的无法解决的 bug。即使出现了 bug 也能很快定位并解决，不会需要大量重构才能解决。

5.2 VGA module

通过学习 VGA 原理的实现，我们发现最重要的操作是信号同步。操作信号同步的消隐期经验数据是我充分借鉴了网上的经验数据得来的，并烧录到板子里实验了一番。

值得一提的是，一开始我忘记了 r，g，b 三个参数应该在 vga_vedio_on 使能信号为 1 时才赋值，因此，一开始整个屏幕是混乱的。注意到这个问题之后使用三目运算符写一个多路选择器：vga_vedio_on ? 0 : data 即可修复。

另一个问题在于，VGA 模块里 RGB 显示是反向的，即 RGB 都是 1111 为白，都是 0000 为黑，所以一开始的颜色是相反的颜色。这个不难解决，在这之前我就写了一个 C 脚本将素材图片中的 RGBA 数据读取出来并生成 IP Core Block Memory Module 的初始化文件.coe 文件，因此我只要在脚本文件中，将数据反向过来输出到.coe 文件，再重新生成 IP Core 即可解决该问题。

Section 6: 经验与总结

这次大作业老实讲花了我不少时间与精力，我总结了以下几点经验：

- 学会使用 IP Core 里面自带的 module，这次我使用了 IP Core 自带的单端口 ROM，简化数据存储和取用部分。如果自己写的话，不仅会导致综合时间过长，也不够优化。
- 使用成熟的设计模式去组织代码，能大量减少思维负担以及 bug 数量。
- Verilog 作为一种硬件语言，要用映射到硬件的思维去组织代码，不能想当然的去写，那样很容易出错。
- 每一个变量要想好是 reg 类型还是 wire 类型，同时数字最好用标准格式，比如 1’b0，不要贪图简便直接写十进制的数字。

最后的总结：我在 Verilog HDL 上实现了一个名为 Flappy Bird 的游戏，它包含了原来 80％的功能，除了开发板之外，我还使用了显示器作为配件。然而，受时间和硬件条件的限制，我的设计还有待改进。