一、项目说明

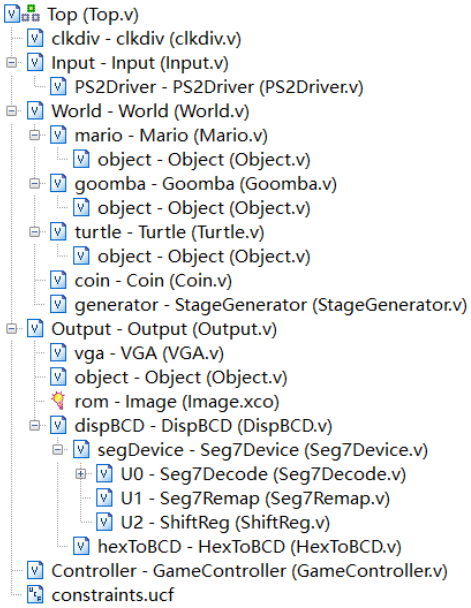
FPGA 超级马里奥游戏使用 PS2 键盘作为输入，VGA、主板七段码作为输出。其中 VGA 用于显示游戏界面，七段码显示游戏所得的分数。PS2 键盘中使用到了左方向键、右方向键、空格键和 R 键。其中方向键用于控制马里奥左右行走，空格键控制跳跃，R 键用于重新开始游戏。游戏分数中，硬币奖励 5 分，升级奖励 20 分，杀死小怪奖励 10 分。

本项目目的在于尽可能还原原版超级马里奥的游戏界面和操作逻辑，此目标已基本实现。游戏包含的要素有马里奥、小怪（goomba）、乌龟（turtle）、砖块、空箱子（box）、未知箱子（包括金钱奖励、升级奖励）、管道（pipe）、硬币（coin）、城堡（castle）、地面（grass）等。游戏动画齐全，逻辑正确，画风还原，交互自然。

二、设计说明

2.1 项目架构

整个项目架构的结构图如下。



项目的核心原理在于将 VGA 扫描坐标信号（col_addr 和 row_addr）映射为 Rom 内存（Image.xco）中的像素点颜色，从而将图片显示在 VGA 上。

下面将从底层模块一步步说明原理，直至最顶层 top 模块。

2.2 Image 模块（Image.xco、Image.v）

Image 模块为项目最底层的模块。该模块由 ISE 的 IP core 工具自动生成，类型为 Block Memory Generator。该 Rom 设计为只读模式，其中内存宽度为 12，深度为 262, 253。读写深度和宽度均相同。内存中按顺序储存游戏用到的所有界面素材，包括元素的所有动画帧，总共有 36 张游戏图片。所有图片采用 12 位色储存方式（用于匹配 VGA 驱动模块的 12 位 vga_data），每个素材都对应着一个 id。

24 位图片原素材已储存在项目根目录下 converter 文件夹中。内存中每个图片素材都拥有一个内存起始位置，该位置记录在 Object 模块中。为了方便生成 Rom 模块的初始化 coe 文件，笔者使用易语言编写了一个转换工具。在将原素材转换为 12 位图片素材的同时，生成了每个素材的起始内存偏移位置，方便之后调用。详见以下的 Object 模块。

值得一提的是原素材的透明色为 F0F，这也是内存中储存的颜色。因此游戏中还需要将该透明色转换为背景蓝色。

2.3 Object 模块（Object.v）

Object 模块将素材的 id 映射为该素材的长度（h），宽度（w）以及内存中的起始偏移量（addr），其中 h 和 w 在游戏核心模块 World 中使用，用于进行碰撞检测、逻辑处理和内存偏移计算。addr 则在输出模块 Output 中用于从内存中读出数据。Object 模块由 2.2 一节所述的转换工具自动生成，它的输入输出定义如下：

```
1. module Object(
2.     input [5:0] id, // 0 - 35 (64)
3.     output reg [10:0] h,
4.     output reg [10:0] w,
5.     output reg [18:0] addr
6. );
```

Object 模块采用 Verilog 行为描述，使用 always @*和 case 语句拟合出组合电路。该模块用于 Mario 模块、Output 模块、Turtle 模块和 Coin 模块。

2.4 Output 模块

Output 模块的作用如下：

1. VGA 处理操作。Output 模块包含 VGA 驱动模块, 输出它的扫描信号 row_addr 和 col_addr, 同时接受该扫描信号对应的屏幕位置参数。该位置参数为 type、h 和 w，分别指该位置所属于的素材 ID、该位置相对于素材的横坐标、相对于素材的纵坐标。Output 模块将根据以上三个信息结合 Object 模块取得该点在内存中的地址。计算的核心代码如下:

```
wire [18:0] addr_begin;
wire [18:0] addr_offset;
wire [10:0] height;
wire [10:0] width;

Object object(
    .id(type),
    .h(height),
    .w(width),
    .addr(addr_begin)
);  // 根据接受的三个位置参数取得内存起始位置

assign addr_offset = addr_begin + h * width + w;  // 计算内存偏移量

Image rom(
    .clka(clkdiv[0]),
    .wea(1'b0),
    .addra(addr_offset),
    .dina(12'b0),
    .douta(rom_data)
);  // 取得内存的 12 位颜色数据

wire [11:0] scene_data;
assign scene_data = (rom_data != 12'hF0F ) ? rom_data : 12'h9CD;//透明色
```

这个设计是考虑到了一个时钟周期只能读 rom 内存一次的特点，因此每当一个时钟上升沿来到时，Output 模块就进行一次映射操作（素材 ID 和素材相对位置 => 内存偏移量）和内存读操作。这样就解决了无法一次性将指定素材的所有数据都读出来的不足。

2. 处理分数数据。这一部分比较简单, 将输入的 num 数据输出至主板七段码驱动模块即可。

3. 处理游戏结束时的黑圈放大缩小操作。该操作主要由下面将说明的 GameController 模块进行。Output 模块接受其传过来的一个 12 位的 mask 参数，该参数只有两种取值：12’h000 和 12’hFFF，分别指该点为黑色、该点不变色。因此只需将 scene_data 和 mask 做一次与操作即可。代码如下:

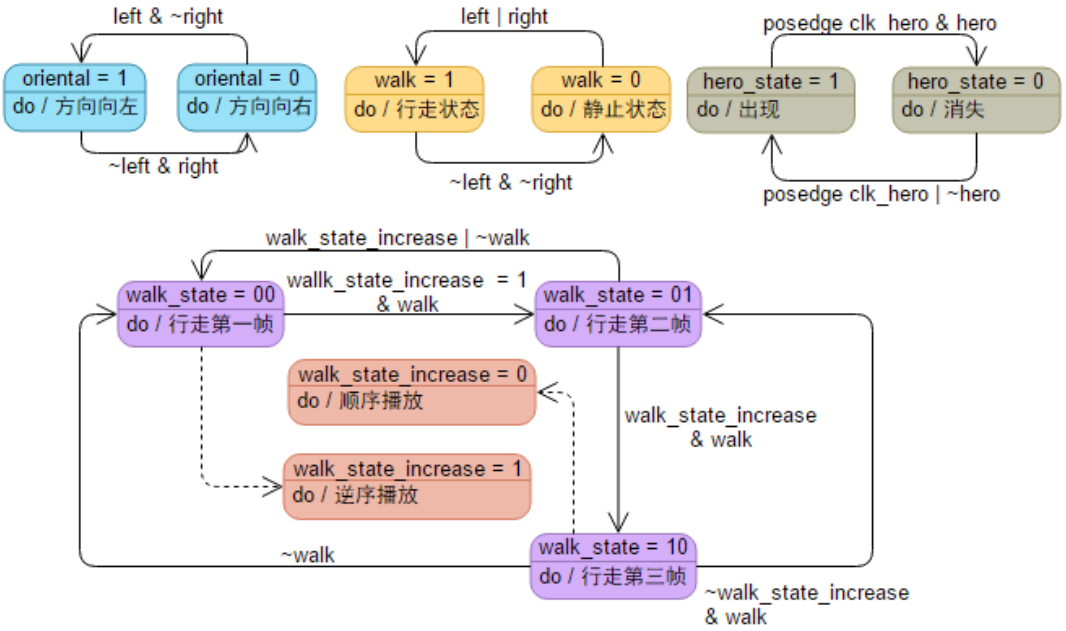
```
1. assign vga_data = mask & scene_data;
```

2.5 游戏元素

该部分将详细介绍游戏的元素。游戏核心为 World 模块以及内部包含的 Mario 模块、Goomba 模块和 Coin 模块。内部包含的三个模块仅仅用于对应游戏元素的内部操作，如逐帧加载动画、方向控制、消隐处理等，不涉及位置移动、碰撞检测等。这些部分由 World 模块进行处理。

2.5.1 Mario 模块（Mario.v）

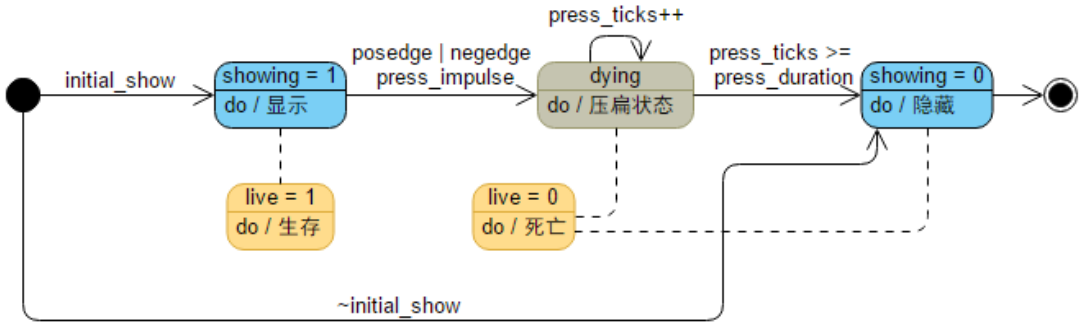
Mario 模块用于控制马里奥。它控制了马里奥的方向（oriental），无敌状态（hero 和 hero_state），等级状态（level），行走帧序号（walk_state）等。Mario 模块的状态转移图如下所示：



以上的 walk_state 状态转移沿时钟 clk_walk_anim 正边沿触发。其他转移若未标明触发条件，皆为 clk 正边沿触发。根据马里奥的方向、行走帧、无敌状态，即可决定此刻马里奥的素材并输出对应的 ID。输出 ID 的过程是未受时钟沿触发控制的组合电路，代码采用 always @*和 case 语句进行编写。Mario 模块同时接入了 Object 模块，输出该 ID 对应的素材宽高至主模块 World，以进行边缘碰撞处理。

2.5.2 Goomba 模块（Goomba.v）

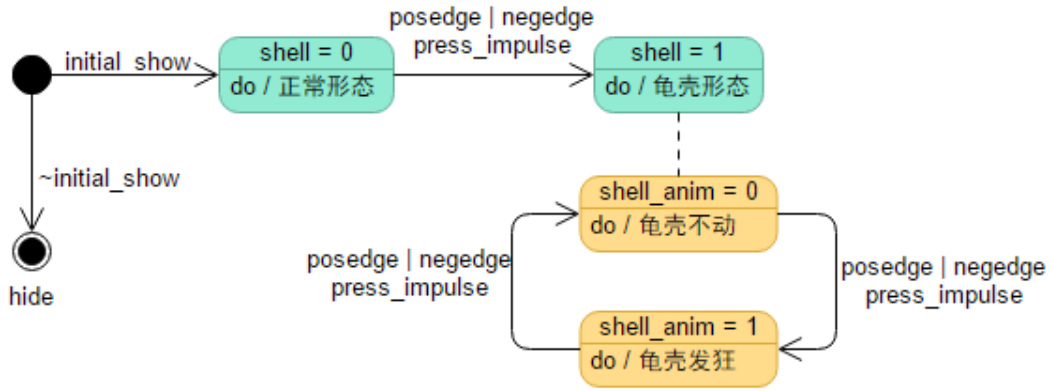
Goomba 模块用于控制小怪，内部状态包含是否显示、是否生存、是否被压扁等。它的状态转移图如下所示：



其中，在 showing 状态至 dying 状态的转移过程里，press_ticks 将初始化为 0，并在之后每一次触发中自增。直至 press_ticks 大于给定值时，小怪才进入消失状态。这就实现了对压扁的控制。live 状态的用途在于判定马里奥受伤条件。只在其 live 状态为真时，马里奥才会受到伤害。图中未标明触发条件的皆为 clk 正边沿触发。此模块同样实现了 Object 模块以输出宽高。

2.5.3 Turtle 模块（Turtle.v）

Turtle 模块用于控制乌龟。乌龟包含的状态有 shell、shell_anim、walk_state 等，其中 walk_state 用于实现乌龟行走时的动画，逻辑和马里奥行走时的动画一致，此处略去不谈。其它参数的状态转移图如下所示：



此模块同样实现了 Object 模块以输出宽高。

2.5.4 Coin 模块（Coin.v）

Turtle 模块用于控制硬币。硬币的动画逻辑与之前的相同，不再赘述。由于硬币所有素材的宽高均为 40，因此此处不需要 Object 模块来输出宽高。

2.5.5 管道（Pipe）、箱子（Box）、地面（Grass）和城堡（Castle）

这四种元素并没有特地新编写了一个模块来处理（这是因为内部状态并不复杂，并未牵涉到动画等复杂的状态转移处理），而是直接写在主模块 World 中。其中值得一提的是 Box 的状态，它的状态转移图如下所示：

（TODO）

2.6 游戏核心逻辑处理（World.v）

（TODO）