VHDL 自主设计实验——俄罗斯方块 + 贪吃蛇

邹卫其 PB16061470

一、实验目的

1、俄罗斯方块

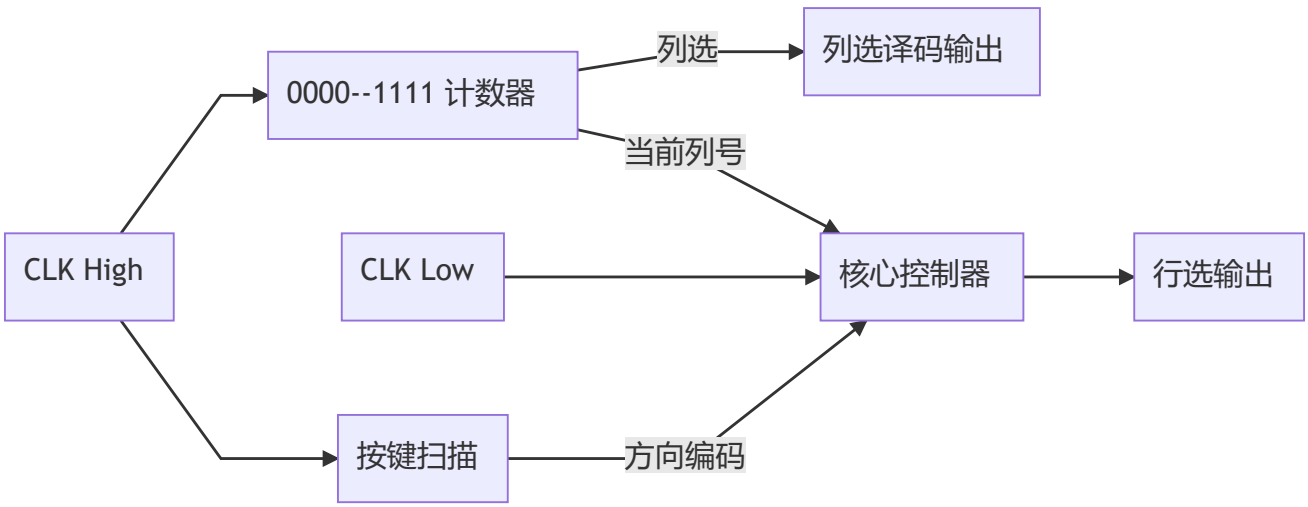
- 基于 8 x 16 （8 x 8 x 2）点阵屏显示
- 方块定时随机产生，每个方块占 4 个格点
- 可控制方块移动，包括左右、下降和旋转
- 满行可相消
- 可 reset 重新开始

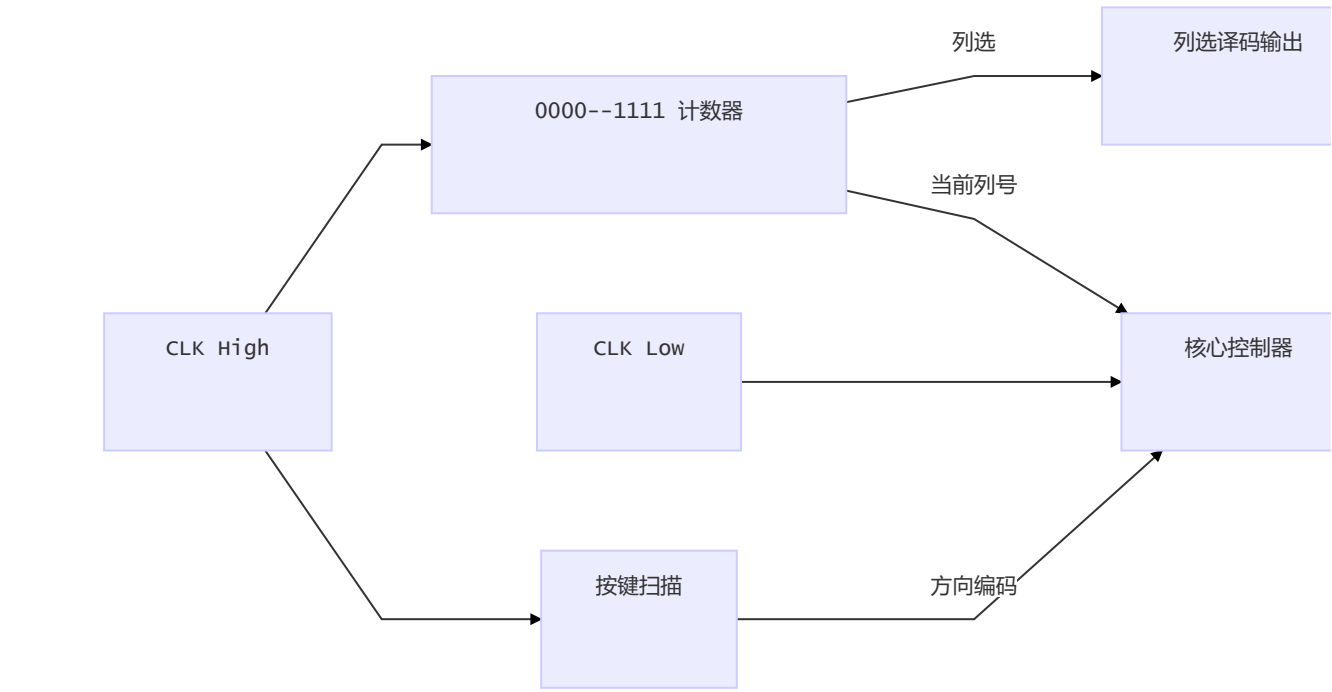
2、贪吃蛇

- 基于 8 x 16 （8 x 8 x 2）点阵屏显示
- 可控制蛇上下左右移动
- 蛇吃每到一颗果实后长度加一
- 一颗果实被吃后，可随机产生新果实
- 可 reset 重新开始

二、设计思路

设计思路框图如下





- **计数**：
  - 由*高速时钟*驱动，产生 0000~1111 循环计数，表示循环的列号
- **列选**
  - 通过计数器*译码输出*，进行 16 列的列选信号输出
- **按键扫描**
  - 由高速时钟驱动，获取四个按键（轻触开关）状态，进行*方向编码*
- **核心控制器**
  - 在*低速时钟*的每个周期内，接收方向编码进行*核心控制*，并锁存游戏数据
  - 锁存 16x8 点阵状态，并根据当前计数值，输出该列号对应的行选信号，完成扫描显示

三、具体实现

1. 俄罗斯方块

数据结构

- **grid**：*signal*，16 * 8 二维数组，储存点阵网格亮灭状态

```
type t_column is array (0 to 7)  of std_logic;   -- grid: (width, height) = (16, 8)
type t_grid   is array (0 to 15) of t_column;
signal grid   :t_grid;
```

- **shapes**：*constant*，28 * 4 * 2 的三维数组，储存所有形状的方块坐标

    - 28 = 7 * 4，包含 7 种基本形状方块，和分别旋转 90、180、270 度对应的其它形状
    - 4 * 2 表示每个方块占四个格点，每个格点有 x、y 二维坐标；此坐标为当方块最贴近左上角时的坐标，代表*基准坐标*

```
type t_position is array (0 to 1)  of integer;   -- position: (x, y), range (0, 0) to (15, 7)
type t_shape    is array (0 to 3)  of t_position;
type t_shapes   is array (0 to 27) of t_shape;
constant shapes :t_shapes := (
        ((0, 0), (0, 1), (1, 0), (1, 1)),   -- 7 basic shapes
        ((0, 0), (1, 0), (2, 0), (3, 0)),
        ((0, 0), (0, 1), (1, 1), (2, 1)),
        ((1, 0), (0, 1), (1, 1), (2, 1)),
        ((2, 0), (0, 1), (1, 1), (2, 1)),
        ((0, 0), (1, 0), (1, 1), (2, 1)),
        ((1, 0), (2, 0), (0, 1), (1, 1)),

        ((0, 0), (0, 1), (1, 0), (1, 1)),   -- rotate 90  deg
        ((0, 0), (0, 1), (0, 2), (0, 3)),
        ((1, 0), (0, 0), (0, 1), (0, 2)),
        ((1, 1), (0, 0), (0, 1), (0, 2)),
        ((1, 2), (0, 0), (0, 1), (0, 2)),
        ((0, 1), (0, 2), (1, 0), (1, 1)),
        ((0, 0), (0, 1), (1, 1), (1, 2)),

        ((0, 0), (0, 1), (1, 0), (1, 1)),   -- rotate 180 deg
        ((0, 0), (1, 0), (2, 0), (3, 0)),
        ((0, 0), (1, 0), (2, 0), (2, 1)),
        ((0, 0), (1, 0), (2, 0), (1, 1)),
        ((0, 0), (1, 0), (2, 0), (0, 1)),
        ((0, 0), (1, 0), (1, 1), (2, 1)),
        ((1, 0), (2, 0), (0, 1), (1, 1)),

        ((0, 0), (0, 1), (1, 0), (1, 1)),   -- rotate 270  deg
        ((0, 0), (0, 1), (0, 2), (0, 3)),
        ((0, 2), (1, 0), (1, 1), (1, 2)),
        ((0, 1), (1, 0), (1, 1), (1, 2)),
        ((0, 0), (1, 0), (1, 1), (1, 2)),
        ((0, 1), (0, 2), (1, 0), (1, 1)),
        ((0, 0), (0, 1), (1, 1), (1, 2))
    );
```

- **current**：*signal*，整型，表示方块形状的下标索引

```
signal current :integer range 0 to 27 := 0;
```

- **dx，dy**：*signal*，整形，表示方块相对于基准坐标的偏移量

```
signal dx    :integer range 0 to 7  := 0;
signal dy    :integer range 0 to 15 := 0;
```

算法设计

方块控制

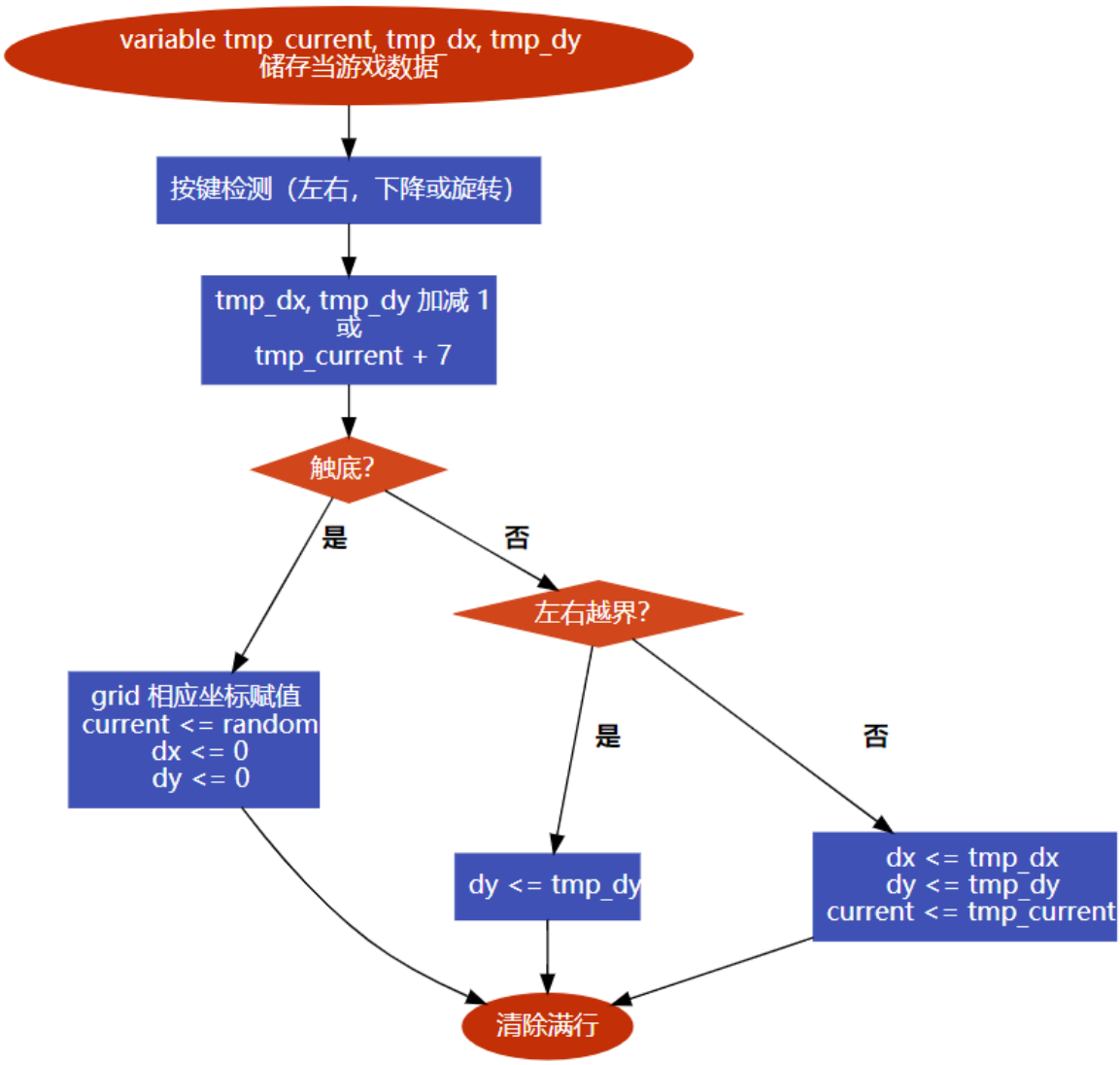
- **索引**：shapes 数组储存了所有方块的形状基准坐标，通过 current 下标值可以确定对应的方块
- **移动**：dx，dy 储存了方块相对于基准坐标的偏移量，每次需要方块四格点坐标时，直接根据 *基准坐标* + *偏移量* 即可算出相应位置；因而，只需要对 dx，dy 进行加减即可轻松控制方块移动
- **旋转**：将方块下标索引值 + 7 即可；原因很简单，shapes 数组存储 28 种所有形状时，先是 7 种基本形状然后对应旋转 90、180、270 度的分别排列，也就是说 current + 7 即对应旋转一次后的下标（当然这里 + 7 是取模意义下的，每旋转 4 次即还原）
- **产生**：只需要产生 0 ~ 27 的随机数即可，因为通过下标即可索引到方块

随机数

- 实现好的随机较难，这里通过高速进程对一个值不停进行计算，其它进程需要的时候取出该值，因为需要取值的时间间隔较为随机，故 random 值较为随机

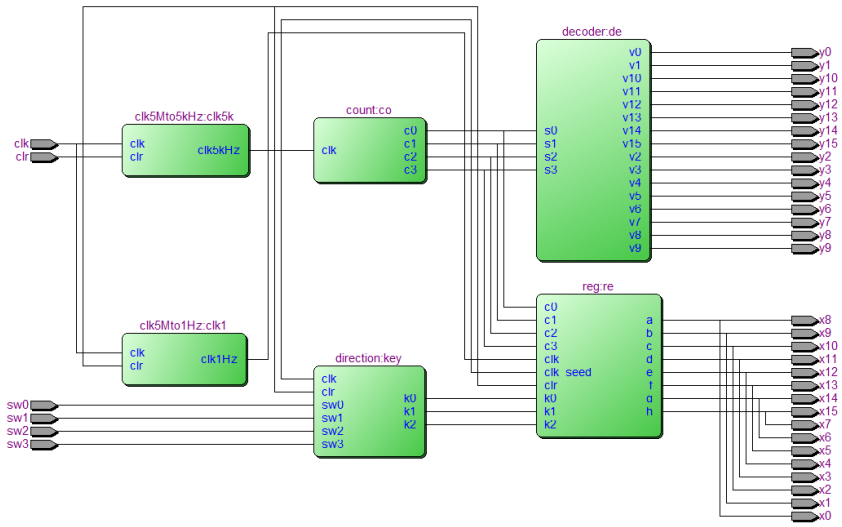
```
seed: process(clk_seed)
begin
    if clk_seed'event and clk_seed = '1' then
        random  <= (random  + 5) rem 28;
    end if;
end process seed;
```

核心控制

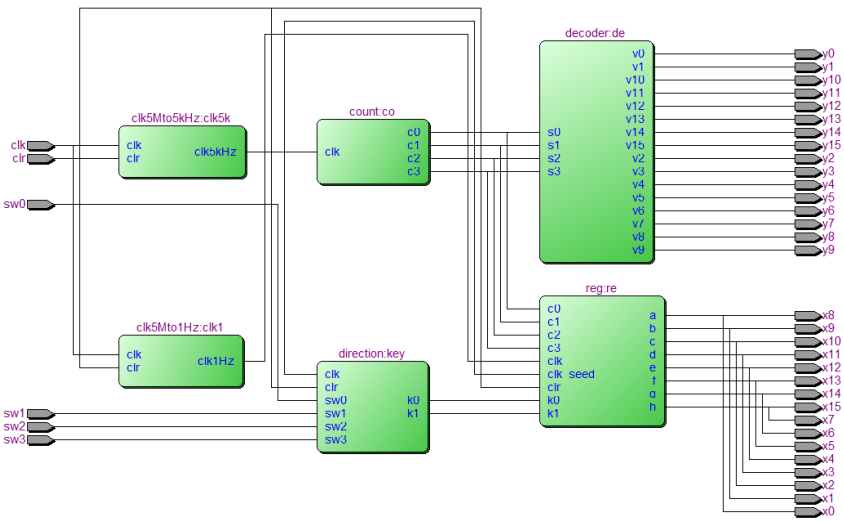


四、RTL 电路

1. 俄罗斯方块

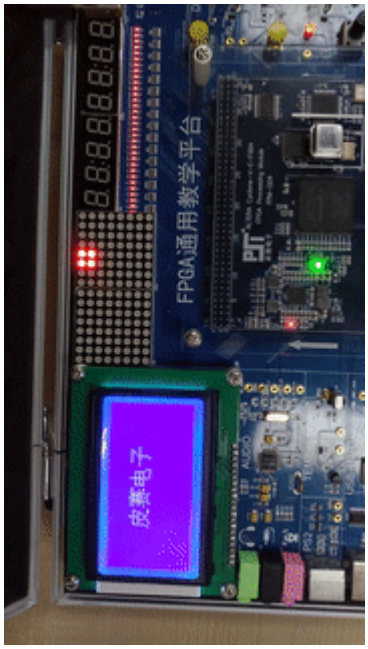


2. 贪吃蛇



五、硬件验证

1. 俄罗斯方块



2. 贪吃蛇