同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2350752

姓　　名 田思宇

专　　业 计算机科学与技术

授课老师 张冬冬

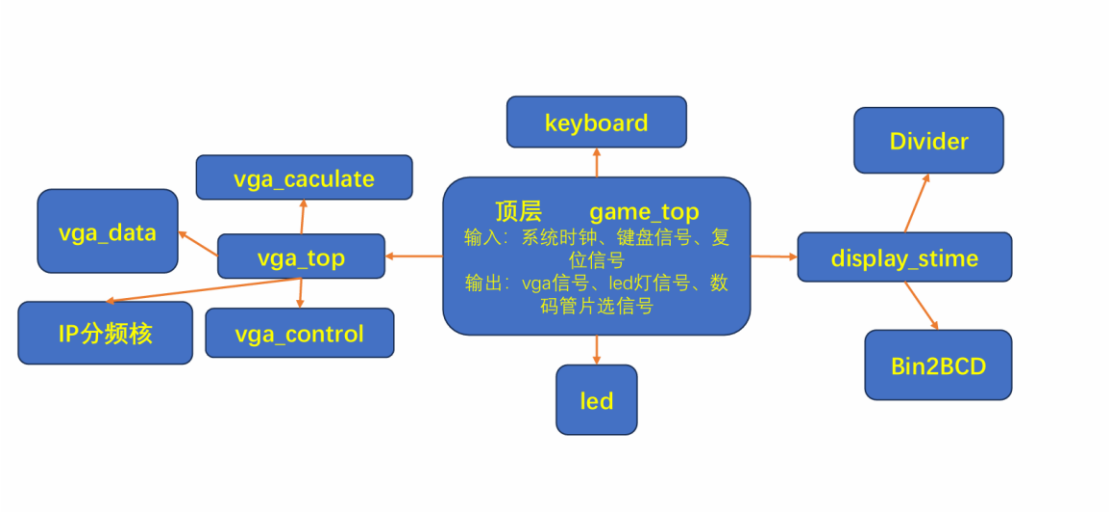
一、实验内容

本项目为基于 VGA 显示器，PS/2 协议键盘设计的一款迷宫小游戏，可以通过键盘上的按钮操作人物移动，并且在 FPGA 板上有时间记录功能(基于七段数码管以及分频器还有 32 位二进制转 BCD 码)，成功走出后，屏幕显示“逃离成功!”字样代表走出迷宫

二、操作说明

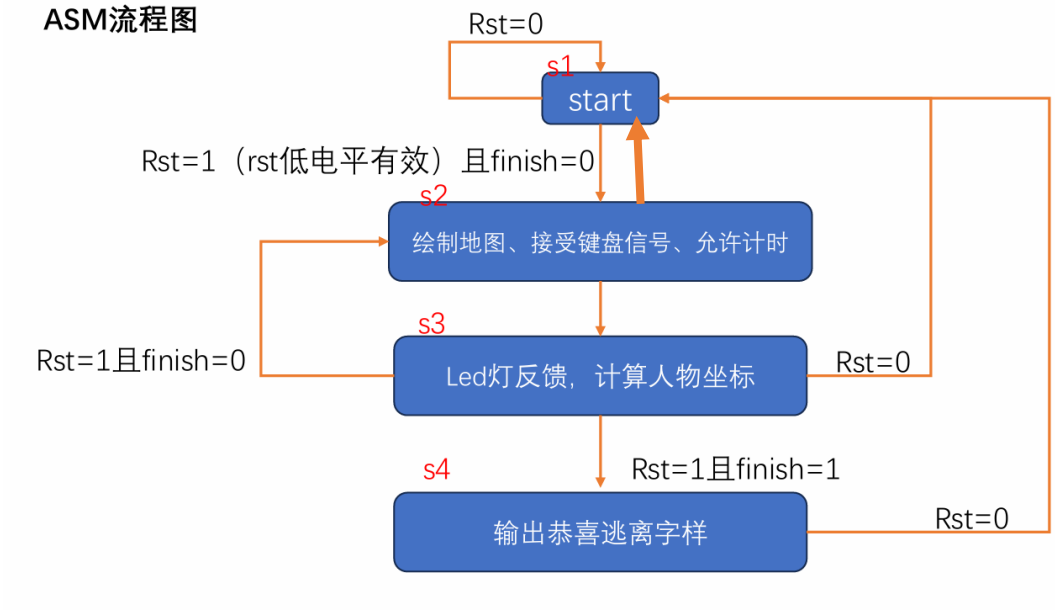
本项目设置了复位输入（J15），低电平有效，如果想要游玩，打开开关 J15 即可，WASD 和上下左右箭头都可以控制移动，空格代表重置，即重新开始游戏。

三、迷宫游戏数字系统总框图



四、系统控制器设计

ASM流程图



下面给状态编码为

S_1：00

S_2：01

S_3：10

S_4：11

控制器的输入有3个

PS A^n	PS B^n	输入 key	输入 rst	输入 finish	NS A^{n+1}	NS B^{n+1}	输出（控制命令）
0	0	X	0	X	0	0	nothing
0	0	X	1	0	0	1	nothing
0	1	X	0	X	0	0	draw、time、receive
0	1	X	1	X	1	0	draw、time、receive
1	0	X	0	X	0	0	Led、caculate
1	0	X	1	0	0	1	Led、caculate
1	0	X	1	1	1	1	Led、caculate
1	1	X	0	X	0	0	congratulation

转移方程：$B^{n+1} = \overline{A^n}\,\overline{B^n}\,rst + A^n\,\overline{B^n}\,rst\,\overline{finish} + A^n\,\overline{B^n}\,rst\,finish$

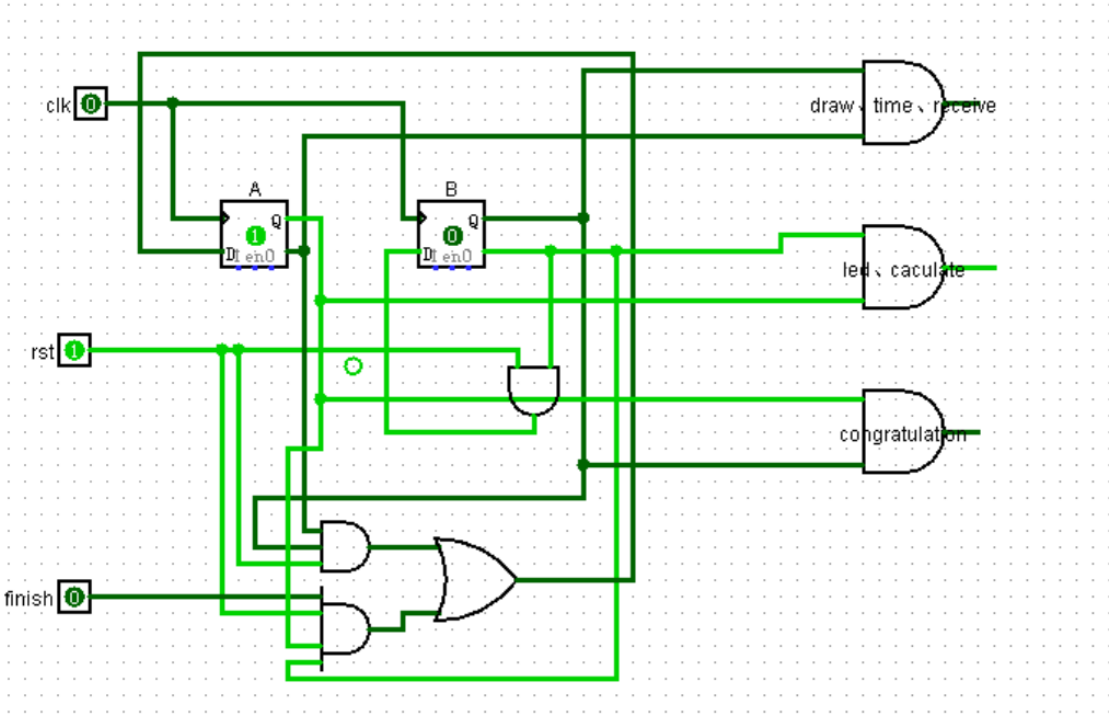
$= \overline{A^n}\,\overline{B^n}\,rst + A^n\,\overline{B^n}\,rst = \overline{B^n}\,rst$

$A^{n+1} = \overline{A^n}\,B^n\,rst + A^n\,\overline{B^n}\,rst\,finish$

控制命令：只与状态有关 $draw = time = receive = \overline{A}\,\overline{B}$

$Led = caculate = A\overline{B}$

$congradulation = AB$

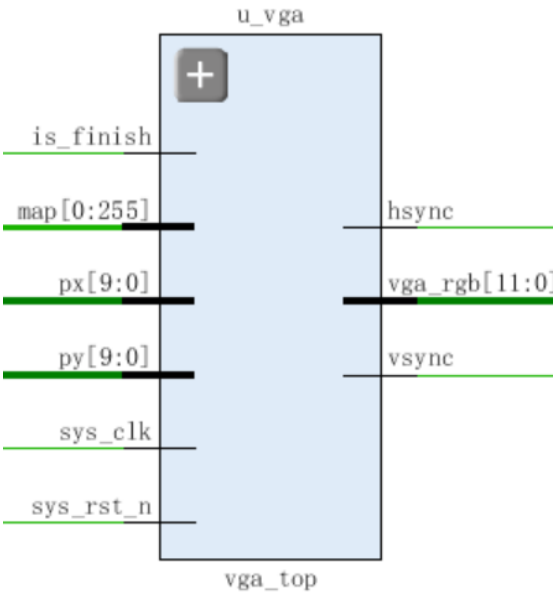


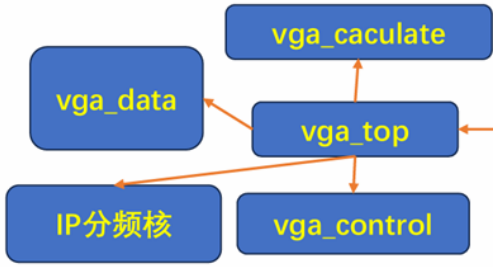
五、子系统模块建模

本系统可划分成四个子系统一共十个子模块，分别是：1、VGA 显示系统（vga_top）：里面包括迷宫逻辑处理模块（vga_caculate）、迷宫颜色处理模块（vga_data）、vga 控制信号处理模块（vga_control）、系统 IP 核分频模块（clk_wiz_0）。2、led 显示系统（led）。时间记录系统：里面包括分频器模块（divider）以及 32 位二进制转 BCD 模块（bin2bcd）。4、键盘处理模块（keyboard）。

1. vga_top

```
module  vga_top
(
    input   wire    [0:255] map         ,//存储地图信息
    input   wire            sys_clk     ,//系统时钟
    input   wire            sys_rst_n   ,//复位信号
    input   wire    [9:0]   px          ,//红点横坐?
    input   wire    [9:0]   py          ,//红点纵坐?
    input   wire            is_finish   ,//是否到达终点标记
    output  wire            hsync       ,//行同步信?
    output  wire            vsync       ,//场同步信?
    output  wire    [11:0]  vga_rgb      //输出三色rgb
);
```





功能描述：本模块是 vga 显示的顶层模块，负责协调四个小模块之间的联系。它接收地图信息、系统时钟、复位信号、红点坐标以及是否到达终点的标记，并输出 VGA 的行同步信号、场同步信号以及 RGB 颜色数据。

设计和实现思路：

时钟分频：使用 clk_wiz_0 IP 核将系统时钟 sys_clk（100MHz）分频为 VGA 时钟 vga_clk（25.175MHz）。。locked 信号用于指示时钟分频是否稳定，只有在时钟稳定后，系统才会解除复位状态。

复位信号处理：

复位信号 rst_n 是系统复位信号 sys_rst_n 和 locked 信号的逻辑与。只有当系统复位信号有效且时钟稳定时，复位信号才会被解除。

VGA 控制模块：

vga_control 模块负责生成 VGA 的行同步信号 hsync 和场同步信号 vsync，并根据当前像素坐标 pix_x 和 pix_y 从 vga_data 模块获取像素数据 pix_data，最终输出 RGB 颜色数据 vga_rgb。

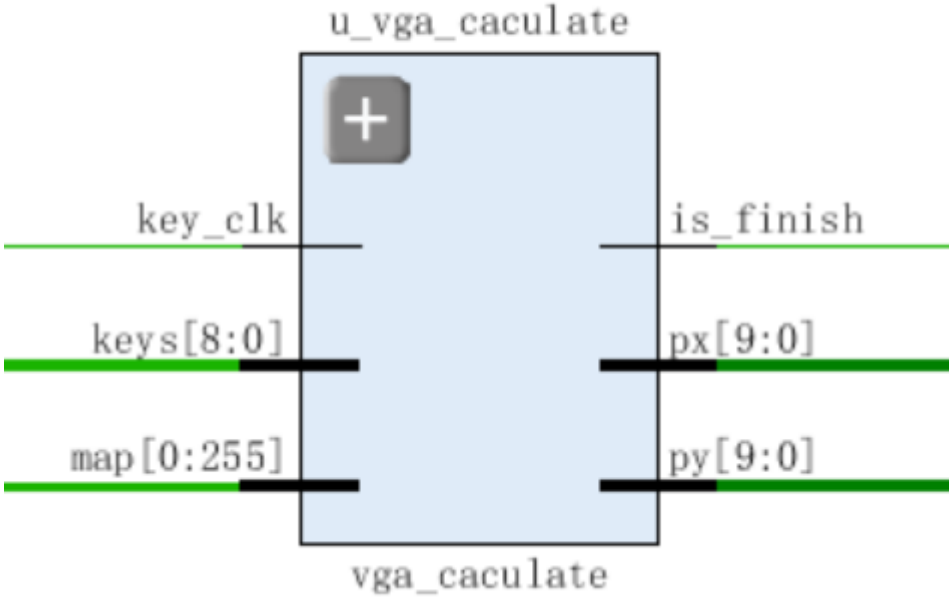
RGB 颜色选择模块：

vga_data 模块根据地图信息 map、当前像素坐标 pix_x 和 pix_y、红点坐标 px 和 py 以及是否到达终点的标志 is_finish，生成当前像素的颜色数据 pix_data。

该模块可能包含逻辑来决定每个像素的颜色，例如根据地图信息显示墙壁、路径，根据红点坐标显示红点，以及根据 is_finish 标志显示终点状态。

2. vga_caculate

```
module vga_caculate(
    input wire[0:255]map,//存储迷宫地图
    input wire key_clk,//键盘时钟
    input wire[8:0]keys,//键盘输入信息
    output [9:0]px,//横坐标
    output [9:0]py,//纵坐标
    output is_finish//判断是否结束的标志
    );
```



功能描述：本模块是 vga 的坐标计算模块，负责处理键盘的输入信号，然后计算出当前红点坐标，同时带有逻辑判断，比如红点坐标在墙的边界就不能再移动了，还有游戏是否结束

设计和实现思路：

红点坐标初始化：

使用寄存器 px1 和 py1 存储红点的当前坐标，初始值分别为 50 和 10。使用寄存器 is_finish1 存储游戏是否结束的标志，初始值为 0。

键盘输入处理：

在 key_clk 的下降沿触发时，根据 keys 的值更新红点的坐标。支持两种键盘输入模式：数字键模式：8'd1（上）、8'd2（下）、8'd3（左）、8'd4（右）。字母键模式：8'h57（W，上）、8'h53（S，下）、8'h41（A，左）、8'h44（D，右）。在移动红点之前，检查目标位置是否为通路（即 map 中对应位置为 0），如果是通路则更新坐标，否则保持坐标不变。

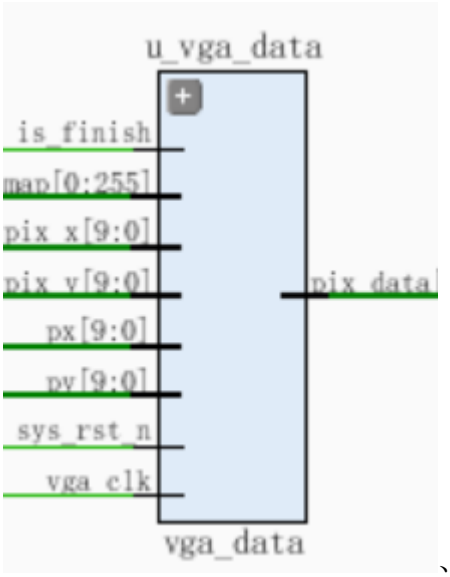
复位功能：

当按下 8'd13（回车键）时，红点坐标重置为初始值 (50, 10)，并将 is_finish1 标志清零。

终点判断：
每次更新红点坐标后，检查红点是否到达终点区域。终点区域定义为横坐标在 13 到 15 之间，且纵坐标大于等于 15。如果红点到达终点区域，则将 is_finish1 标志置为 1。

3. vga_data

```
module  vga_data
(
    input   wire [0:255]map,//存储地图信息
    input   wire vga_clk,//65MHZ时钟
    input   wire sys_rst_n,//复位信号
    input   wire [9:0] pix_x,//当前像素横坐标
    input   wire [9:0] pix_y,//当前像素纵坐标
    input   wire [9:0] px,//红点横坐标
    input   wire [9:0] py,//红点纵坐标
    input   wire is_finish,//判断是否结束的标志
    output  reg [11:0] pix_data//当前像素点的三色rgb信息
);
```



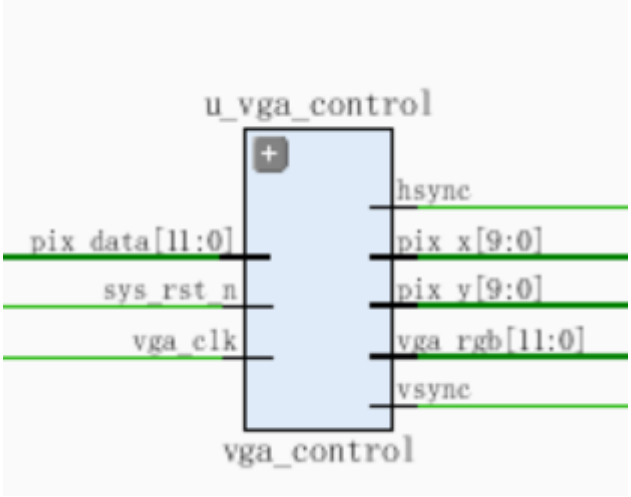
、

功能描述：本模块是 vga 颜色数据处理模块，根据地图数据、游戏结束后显示的点阵数据、当前红点坐标来决定 rgb 的信号。
实现思路：
颜色定义：使用 parameter 定义了多种颜色，包括红色 (RED)、黄色 (YELLOW)、青色 (CYAN)、蓝色 (BLUE)、紫色 (PURPLE)、黑色 (BLACK) 和白色 (WHITE)。
字模定义：使用 reg 定义了一个 1280 位的字模 congrats_msg，用于存储“逃离成功!”的显示信息。每个位代表一个像素点（1 表示显示，0 表示不显示）。
像素颜色生成逻辑：在 vga_clk 的上升沿触发时，根据当前像素坐标 (pix_x, pix_y) 和其他输入信号生成像素颜色数据 (pix_data)。
如果游戏未结束 (is_finish == 0)：如果复位信号有效 (sys_rst_n == 0)，则将像素颜色设置为紫色 (PURPLE)。如果当前像素位于红点附近（pix_x 和 pix_y 在红点坐标 (px, py)

的 ±2 范围内），则将像素颜色设置为紫色 (PURPLE)。如果当前像素位于墙壁区域（map 中对应位置为 1），则将像素颜色设置为黄色 (YELLOW)。否则，将像素颜色设置为白色 (WHITE)。如果游戏结束 (is_finish == 1)：如果当前像素位于“逃离成功!”字模的显示区域（pix_x 在 50 到 450 之间，pix_y 在 100 到 180 之间），并且字模中对应位置为 1，则将像素颜色设置为紫色 (PURPLE)。否则，将像素颜色设置为白色 (WHITE)。

4. vga_control

```
module  vga_control
(
    input   wire            vga_clk     ,//25MHZ时钟
    input   wire            sys_rst_n   ,//复位信号
    input   wire    [11:0]  pix_data    ,//当前像素的rgb数据
    output  wire    [9:0]   pix_x       ,//当前像素横坐标
    output  wire    [9:0]   pix_y       ,//当前像素纵坐标
    output  wire            hsync       ,//行同步信号
    output  wire            vsync       ,//场同步信号
    output  wire    [11:0]  vga_rgb      //输出三色rgb
);
```



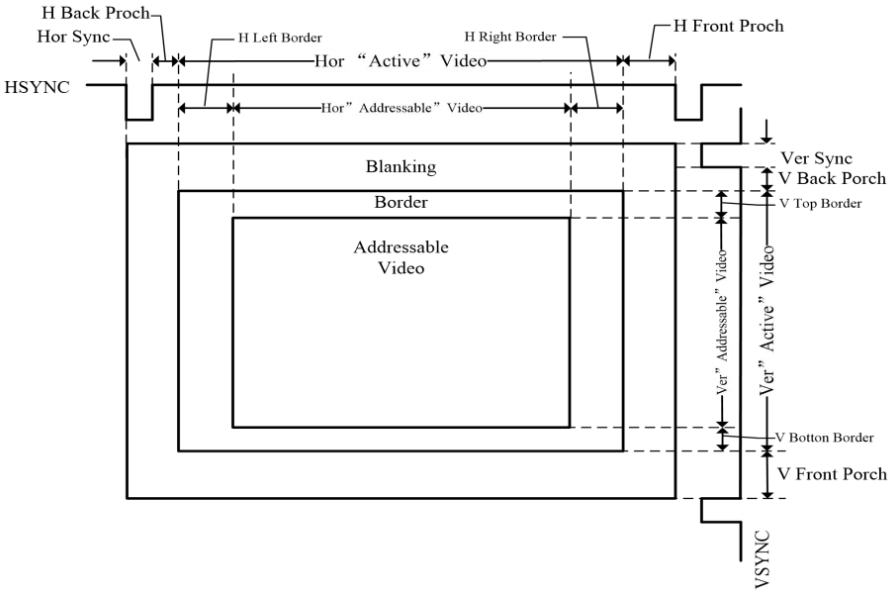
功能描述：这是 vgn 子系统中最重要的一个模块，通过 25MHZ 的时钟得到行同步信号 hsync 与场同步信号 vsync，并且输出每一个像素点的 rgb 颜色信号。这是 vgn 显示器能否显示的关键

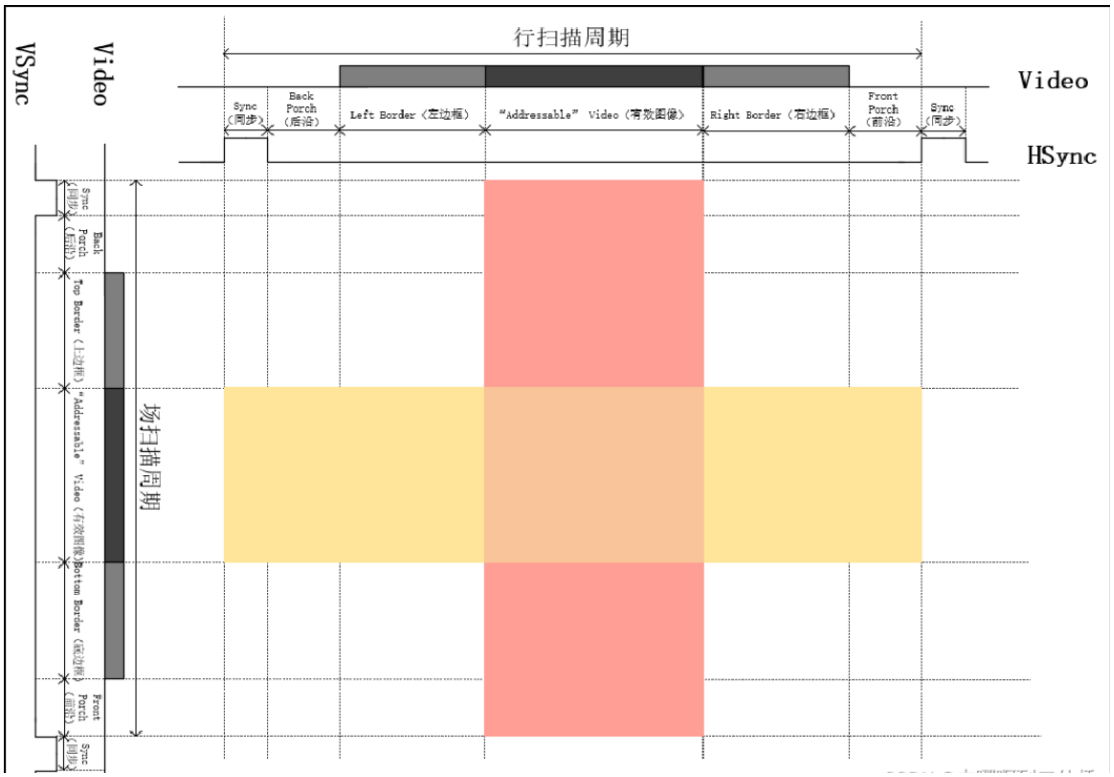
VGN 实现思路（工作原理）：

我们都知道显示器或者图片都是由像素点构成的，所以想让显示器显示出图片，只需要让显示器一点一点的把像素点输出出来就可以了。而 vga 显示器的扫描方式就是逐行扫描，先扫描一行然后移动到下一行，工作原理非常像打字机。

VGA 的时序原理：vga 的时序分为行时序和场时序，一个完整的行扫描周期，包含 6 部分： Sync （同步）、 Back Porch （后沿）、 Left Border（左边框）、“Addressable” Video （有效图像）、 Right Border （右边框）、 Front Porch（前沿）。一个完整的场扫描周期，也包含 6 部分： Sync （同步）、 Back Porch （后沿）、 Top Border（上边框）、“Addressable” Video （有效图像）、 Bottom Border （底边框）、 Front Porch（前沿），

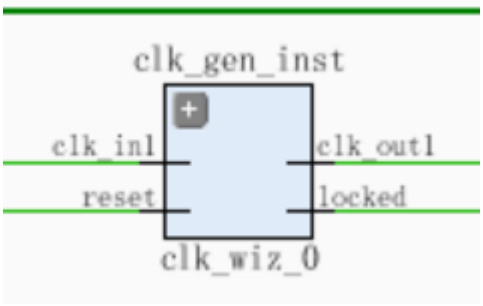
与行同步信号不同的是，这 6 部分的基本单位是 line （行），即一个完整的行扫描周期。在一个完整的场扫描周期中，Video 图像信息在 HSync（行同步信号）和 VSync（场同步信号）的共同作用下完成一帧图像的显示，Video 图像信息只有在 “Addressable” Video（有效图像）阶段，图像信息有效，其他阶段图像信息无效。VSync 行同步信号在 Sync （同步）阶段，维持高电平，其他阶段均保持低电平，完成一个场扫描周期后，进入下一帧图像的扫描。所以只有在下图两个颜色重合的部分，才能输出有效的颜色信号。了解了这些原理之后，设计出这个模块就很简单了，本模块使用了两个寄存器，分别记录当前 vgn 显示器扫描点的行坐标和列坐标，然后根据 vgn 工作原理计算出是否是有效输出位置，同时计算出像素的 x、y 坐标、同时产生行同步和列同步信号、以及如果不在有效区域内，设置 rgb 为黑色。





5. clk_wiz_0

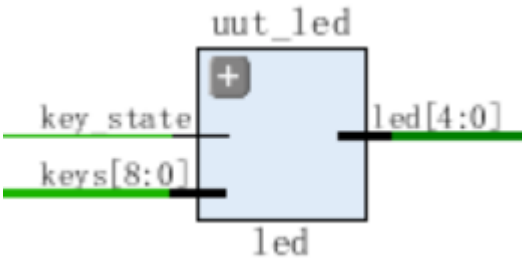
```
clk_wiz_0 clk_gen_inst
(
    .clk_in1    (sys_clk),      // 输入系统时钟100mhz
    .reset      (~sys_rst_n),   // 输入复位信号
    .clk_out1   (vga_clk),      // 得到降频后的时钟信号25.175mhz
    .locked     (locked)        // 得到Locked信号
);
```



描述：本部分很简单，调用自带的 ip 核实现了把系统 100mhz 的 clk 分频成 25.175mhz，这是因为我选择的显示参数是 640*480 的，所以必须把 vgn 的时钟设置成 25.175mhz 才可以用，其实这里有一个计算公式，便是 800*525*60≈25.175mhz。（为什么选择 640*480 会在后面心得体会中提到）

6. led

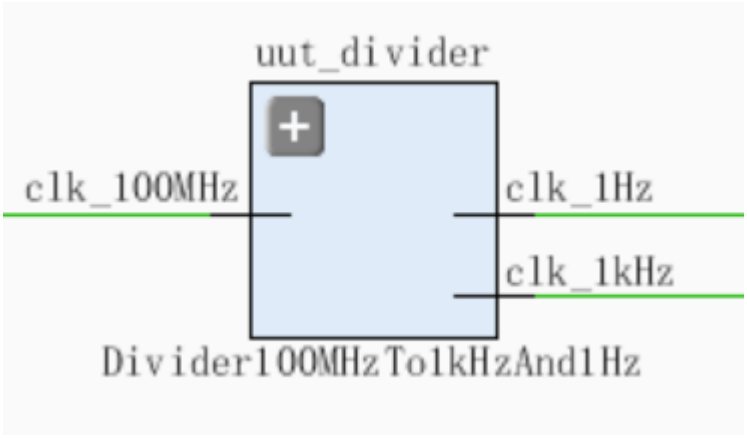
```
module led(
    input key_state     ,//键盘按键状态
    input [8:0]keys     ,//按下键盘的ASCII值
    output reg [4:0] led //控制哪个led灯亮起,分别代表前后左右以及复位（）
);
```



描述：本模块的实现同样也比较简单，根据键盘的状态使用 always 语句，判断 keys 的 ascii 值，从而确定要亮哪一个灯，在亮灯之前对 led 数组进行了初始化，这样就保证了每次只会亮起一个 led 灯，写这个模块的目的主要是用来直观的判断键盘按键按下去是否奏效了。

7. divider

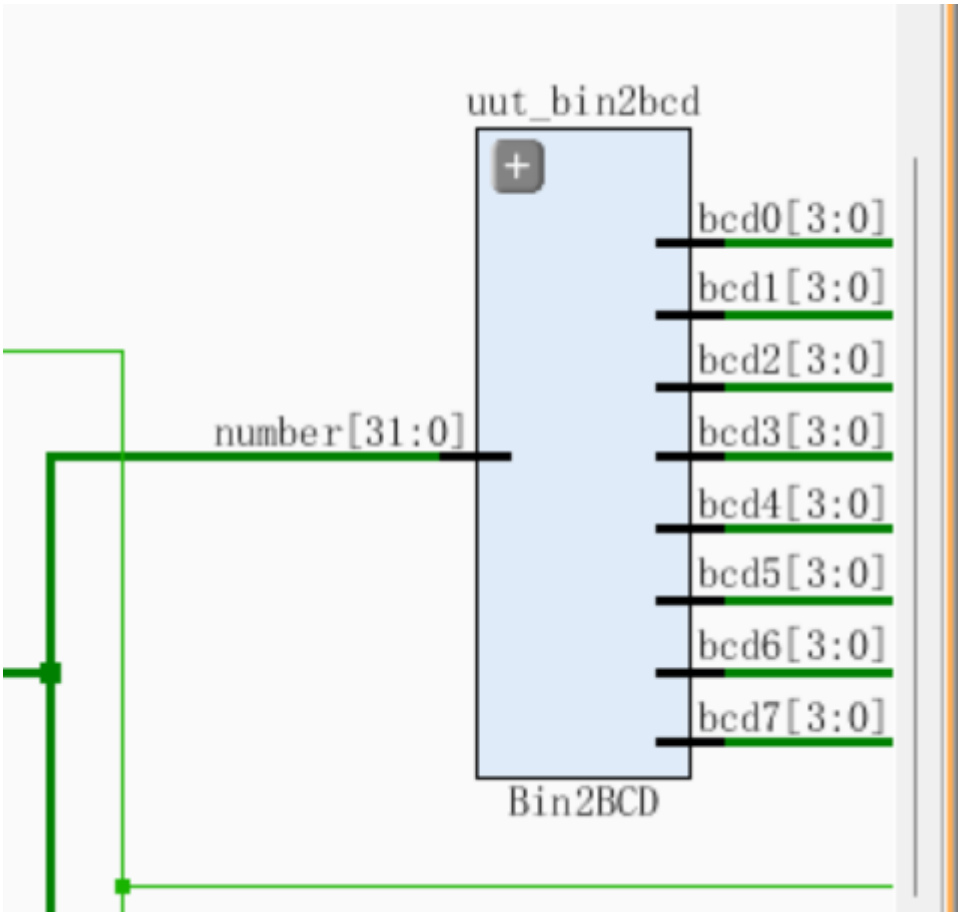
```
module Divider100MHzTo1kHzAnd1Hz(
    input wire clk_100MHz,  // 输入100MHz时钟信号
    output reg clk_1kHz = 0,    // 输出1kHz时钟信号
    output reg clk_1Hz = 0      // 输出1Hz时钟信号
);
```



描述：本模块主要是给 displaytime 模块打辅助的，因为系统的时钟是 100Mhz，但是 ip 核又无法分频到 1000hz 或者 1hz，所以只好手动写一个分频器模块，以前实验课上写过 1hz 分频器，这里稍微改进了一下，让一个模块可以有双分频作用，最终分频出 1hz 和 1000hz 的时钟，其中 1hz 时钟用于记录时间，1000hz 时钟用于刷新七段数码管（为什么选择 1000hz 会在后面心得体会中提到）

8. bin2bcd

```
module Bin2BCD(
    input   [31:0]      number,
    output [3:0] bcd0,
    output [3:0] bcd1,
    output [3:0] bcd2,
    output [3:0] bcd3,
    output [3:0] bcd4,
    output [3:0] bcd5,
    output [3:0] bcd6,
    output [3:0] bcd7
    );
```



描述：本模块主要是把 32 位二进制的时间转化成 8 个 bcd 码方便数码管单独输出。

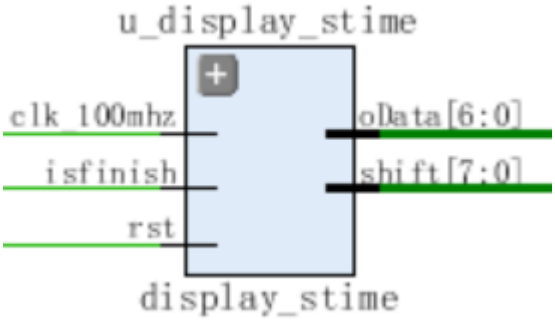
设计和实现思路：

双移位加 3 算法：该算法通过逐位左移和条件加 3 操作，将二进制数转换为 BCD 码。每次左移后，检查每个 4 位 BCD 码是否大于 4，如果是，则加 3。重复上述操作，直到所有二进制位都被处理。

实现步骤：初始化一个 32 位的寄存器 result，用于存储中间结果。将输入的二进制数 number 赋值给 bin。使用 repeat 循环进行 31 次移位和加 3 操作。最后一次移位后，将最高位赋值给 result[0]。最终将 result 的每 4 位分配给对应的 BCD 输出。

9. displaytime

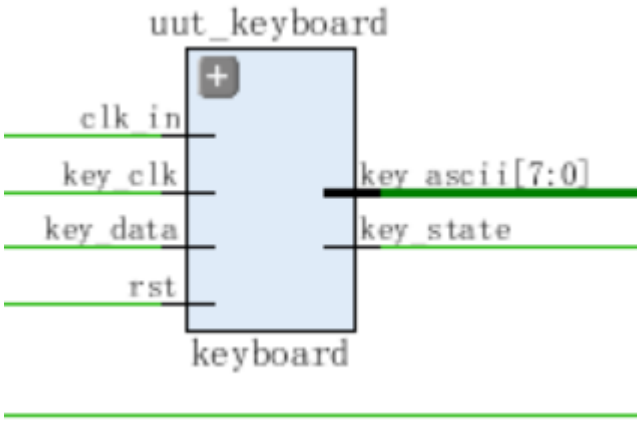
```
module display_stime(
    input rst,
    input isfinish,
    input clk_100mhz,
    output reg [7:0] shift = 8'b1111_1111, // 数码管片选信号
    output reg [6:0] oData = 7'b1111111    // 数码管段选信号
);
```



描述：本模块主要是实例化上面两个小模块，帮助处理数据，最后自己根据这些数据在七段数码管中显示数据，但是这里有改进的地方（不是简单照搬实验课的，而是对实验课的改进提高），实验课上做的七段数码管显示八个数码管显示的都是一个数据，但是这里八个数码管显示不同的数据，主要的实现方法就是设置一个 cnt 数组，根据前面产生的 8 个 bcd 码，采用循环的方式（每次 cnt 数组+1），对七段数码管片选信号更新、以及七段数码管数据更新。虽然一次只在八个七段数码管中显示了一个数码管，但是根据人眼的视觉残留效应，看起来像是同时显示了八个七段数码管。

10. keyboard

```
module keyboard//将键盘键码转化为ASCII码
(
input                   clk_in,             //系统时钟
input                   rst,                //系统复位，低有效
input                   key_clk,            //PS2键盘时钟输入
input                   key_data,           //PS2键盘数据输入
output  reg             key_state,          //键盘的按下状态，按下置1，松开置0
output  reg     [7:0]   key_ascii           //按键键值对应的ASCII编码
);
```



描述：本模块主要处理键盘的输入信号，并且把他们转化成对应的 ascii 码输出到其他模块中，这里设置了 wasd 和方向键来进行前后左右的移动，空格实现游戏重置，一共九个键，采用 case 语句区分，其他按键按下去都无效。

PS/2 协议原理：

PS/2 协议采用双向同步串行通信，通信由主机（计算机）或从设备（键盘/鼠标）发起。通

信的基本单位是一个 11 位的帧，包括：1 位起始位（0）。8 位数据位（从低位到高位传输）。1 位奇偶校验位。1 位停止位（1）。PS/2 型键盘主要使用扫描码来进行数据传输。每个按键被按下时都会被分配固定的扫描码，如果按键一直被按下，那么该按键码值将每隔 100ms 重复发送一次。当按键释放后，一个关键码 F0 发送后，该扫描码才会被发送。当使用上档键进行组合键时，也会产生上档键的码值，并且由主机决定到底是哪一个 ASCII 值。当组合键被释放后，一个 E0 和 F0 关键码发送后，按键扫描码才被发送。所以获得按键的扫描码尤为重要，本模块也就是根据处理后得到扫描码转化成我们常见的 ascii 码，再传输给其他模块的。

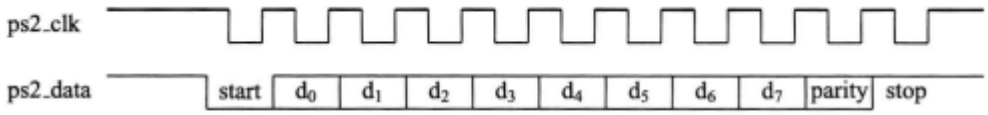
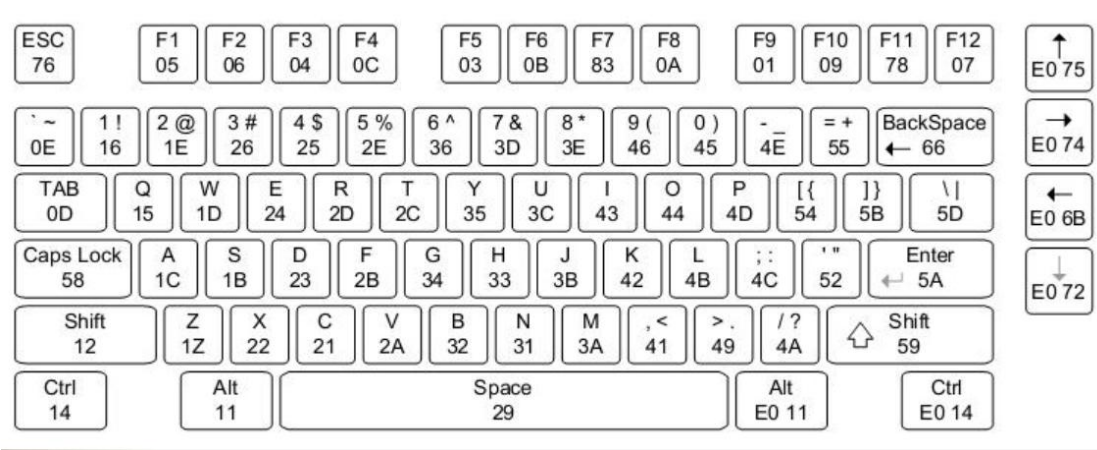
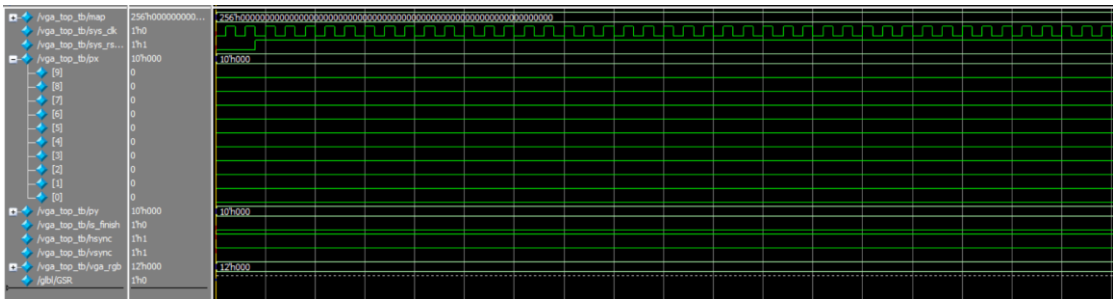


图 15.14　PS/2 键盘送数据给主机的时序图



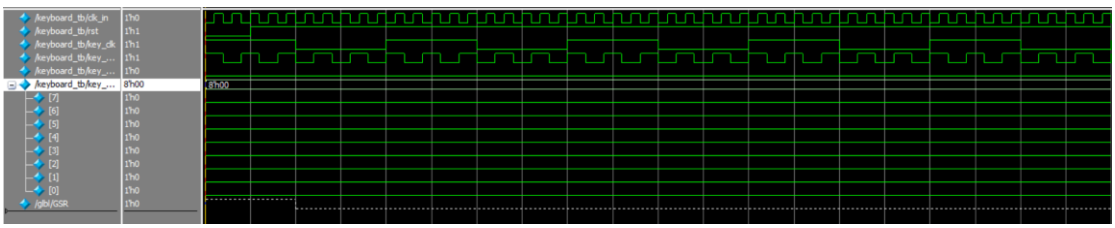
六、测试模块建模

1. vga_top_tb



主要模拟了 vga 时序的输出是否正确。

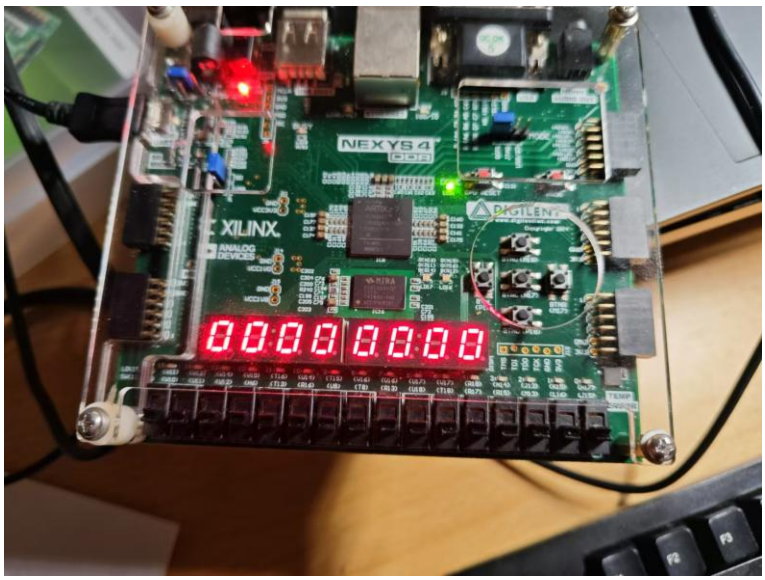
keyboard_tb



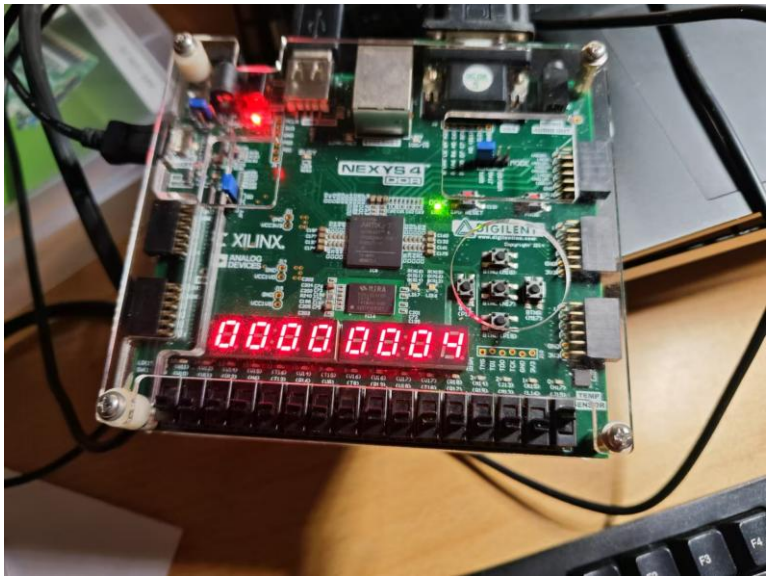
主要模拟了系统复位，clk 时钟输入、clk 数据时钟三个功能，波形图验证了模块功能正常。

七、实验结果

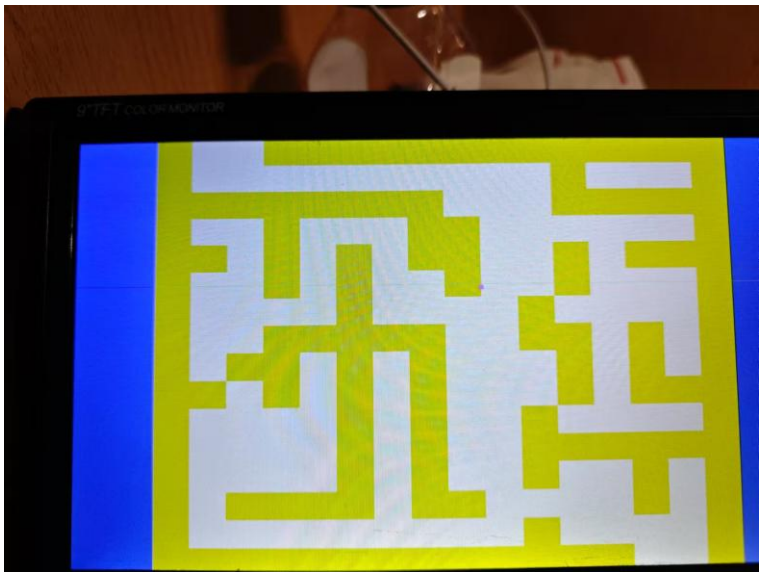
刚刚下板后，由于 rst 信号是低电平，所以不记录时间，屏幕也没有内容显示



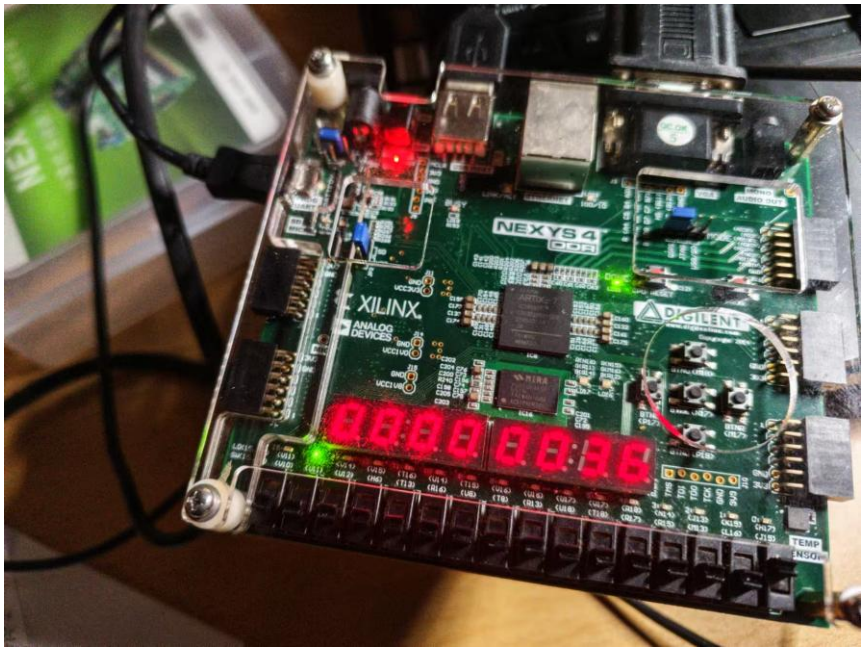
打开最右边开关，rst 变成高电平，此时开始记录游戏时间，同时屏幕显示游戏内容



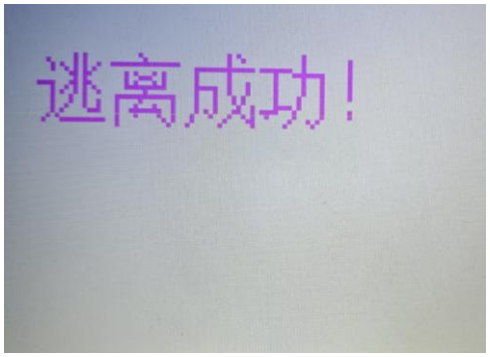
进入游戏界面后通过键盘控制移动，如下图移动到边界后再次按下键盘向左但是并不会移动



这里按下键盘后对应的 led 灯亮起说明 led 灯模块正常工作



成功到终点后会显示逃离成功！同时记时停止。



八、结论

1. 成功跑通了两个带协议的模块（vga 和 ps/2）。
2. 实现了时序的合理安排，各模块之间时序没有冲突。
3. 开发了计时功能，提高了游戏的游玩性。
4. 在开发周边功能时，进一步对课上的实验进行了拓展，比如一个分频器两用，再比如七段数码管单独显示功能、以及课上没教过的 32 位 2 进制转 bcd 码的方法。
5. 深入了解了 vga 和 ps/2 的工作原理，知道以后如何在 vga 上显示数据，如何处理 ps/2 输入的数据。

九、心得体会及建议

1、 首先是自顶向下这四个字：在刚开始设计的时候，不知道要设计什么样的游戏，最后好不容易确定下来设计一个迷宫之后，我反而直接开工 vga 时序同步信号处理的模块，这让我倍感痛苦，寸步难行，一度想要放弃，后来仔细思考一番后，看看数字逻辑书上的案例，我开始在草稿纸上规划要实现的功能，然后规划实现这些功能需要哪些模块，然后再慢慢想这些模块之间是怎么通信的？最后把之前一团糊的设计全部删掉从头开始做。这件事让我深刻认知到自顶向下的重要性，一定要再草稿纸上规划好，然后再具体做，否则很容易陷入死胡同，同时多模块程序，模块之间的通信问题也很重要，在设计的时候我设置变量有点小糊涂，虽然整体规划了一些输入输出，但是具体的一些变量还是没能考虑到，在写模块的时候只好发现哪里缺了就在哪里补上。所以如何找到一个通用方法设计所需要的变量？我想这是我接下来学习过程中必须要掌握的。

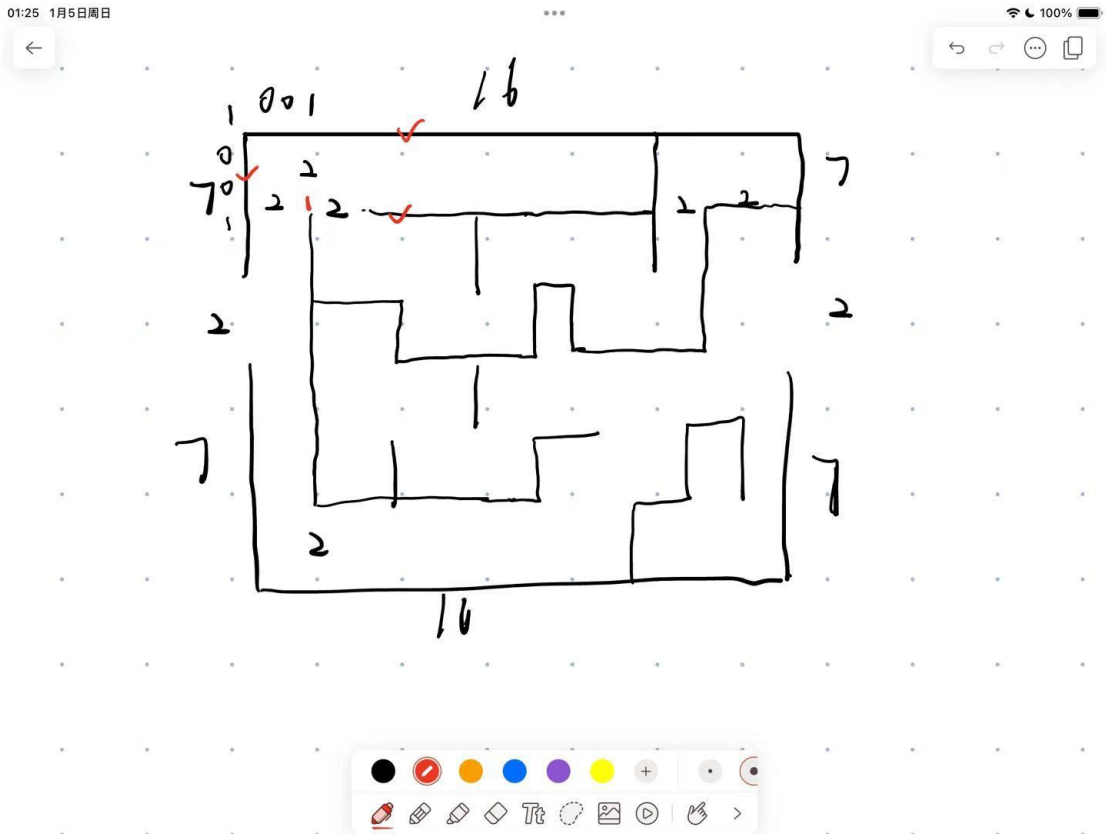
2、 模块之间的串联逻辑有很多，如何选择串联逻辑让代码更加简洁易懂？我想也是我以后需要掌握的。

3、 关于 vga 的参数选择问题，我的 vga 的分辨率是 1024*768 的，但是我根据参数表配置好 1024*768 的设置之后（分频是 65Mhz），能够产生 bit 流，但是下板之后 vga 就显示信号不支持（如下图），当时检查不出到底是哪里出了问题，测试文件跑下来也都是正常的，正当我一筹莫展的时候，同学告诉我不一定要设置成 1024*768 的，设置成小一点的也可以，只要是 4：3 的比例，然后控制好地图每个元素大小。于是我改用了 640*480 的来显示，同时把 IP 核改成 25.175Mhz，最后成功在 vga 中显示。

4、 关于在 vga 上显示图片的心得体会：在了解完 vga 是一个像素点一个像素点打印的时候，由于我设计的是迷宫游戏，有许多像素点是重复的，如果我用 640*480 大小的 rom 去存储一个地图显然太浪费了，所以我设计了一个 16*16 的地图，然后采用等比例缩放的方法，也就是地图上每一个块代表 40*30 的像素块，这样就大大节省了存储空间，以及迷宫的设计时间（迷宫是自己在 ipad 手画出来，然后计算出对应的二进制代码的）见下图。

5、 七段数码管显示：我设计了一个记录时间的模块，一开始想搬用课上学到的七段数码管代码，但是那个代码只能显示 0-9 数字，这显然不合理，所以只好自己重新思考，得出结论是需要单独隔离开 8 个数码管，正好利用片选信号，然后学习了一下二进制转 bcd 的算法于是设计出了对应的模块

信号不支持
POWER
+
M
-
V1/V2
MODE
AV IN
PHONE



十、附录

设计文件：

```
module Bin2BCD(
    input     [31:0]          number,
    output [3:0] bcd0,
    output [3:0] bcd1,
    output [3:0] bcd2,
    output [3:0] bcd3,
    output [3:0] bcd4,
    output [3:0] bcd5,
    output [3:0] bcd6,
    output [3:0] bcd7
    );

    reg [31:0]    bin;
    reg [31:0]    result;
    reg [31:0]    bcd;
    //t□□ÏBCD□□
    always @(number) begin
        bin = number[31:0];
        result = 32'd0;
        repeat (31)
```

```
        begin
            result[0] = bin[31];
            if (result[3:0] > 4)
                result[3:0] = result[3:0] + 4'd3;
            else
                result[3:0] = result[3:0];
            if (result[7:4] > 4)
                result[7:4] = result[7:4] + 4'd3;
            else
                result[7:4] = result[7:4];
            if (result[11:8] > 4)
                result[11:8] = result[11:8] + 4'd3;
            else
                result[11:8] = result[11:8];
            if (result[15:12] > 4)
                result[15:12] = result[15:12] + 4'd3;
            else
                result[15:12] = result[15:12];
            if (result[19:16] > 4)
                result[19:16] = result[19:16] + 4'd3;
            else
                result[19:16] = result[19:16];

            if (result[23:20] > 4)
                result[23:20] = result[23:20] + 4'd3;
            else
                result[23:20] = result[23:20];

            if (result[27:24] > 4)
                result[27:24] = result[27:24] + 4'd3;
            else
                result[27:24] = result[27:24];
            if (result[31:28] > 4)
                result[31:28] = result[31:28] + 4'd3;
            else
                result[31:28] = result[31:28];
            result = result << 1;
            bin = bin << 1;
        end
        result[0] = bin[31];
        bcd = result;
    end

    assign bcd0 = bcd[3:0];
```

```
    assign bcd1 = bcd[7:4];
    assign bcd2 = bcd[11:8];
    assign bcd3 = bcd[15:12];
    assign bcd4 = bcd[19:16];
    assign bcd5 = bcd[23:20];
    assign bcd6 = bcd[27:24];
    assign bcd7 = bcd[31:28];
endmodule

module Divider100MHzTo1kHzAnd1Hz(
    input wire clk_100MHz,    // 输入 100MHz 时钟信号
    output reg clk_1kHz = 0,      // 输出 1kHz 时钟信号
    output reg clk_1Hz = 0          // 输出 1Hz 时钟信号
);

    // 分频计数器
    reg [16:0] count_1kHz = 0;   // 计数器用于 1kHz 分频
    reg [26:0] count_1Hz = 0;    // 计数器用于 1Hz 分频

    // 1kHz 分频
    always @(posedge clk_100MHz) begin
        if (count_1kHz == 49999) begin
            count_1kHz <= 0;
            clk_1kHz <= ~clk_1kHz;
        end else begin
            count_1kHz <= count_1kHz + 1;
        end
    end

    // 1Hz 分频
    always @(posedge clk_100MHz) begin
        if (count_1Hz == 49999999) begin
            count_1Hz <= 0;
            clk_1Hz <= ~clk_1Hz;
        end else begin
            count_1Hz <= count_1Hz + 1;
        end
    end

endmodule

module display_stime(
    input rst,
    input isfinish,
```

```
    input clk_100mhz,
    output reg [7:0] shift = 8'b1111_1111, //  数码管片选信号
    output reg [6:0] oData = 7'b1111111     //  数码管段选信号
);
    reg [31:0] stime = 0;
    wire [3:0] Data[7:0];
    reg [3:0] cnt = 0;
    wire clk_1000hz;
    wire clk_1hz;

    Bin2BCD uut_bin2bcd(
        .number(stime),
        .bcd0(Data[0]),
        .bcd1(Data[1]),
        .bcd2(Data[2]),
        .bcd3(Data[3]),
        .bcd4(Data[4]),
        .bcd5(Data[5]),
        .bcd6(Data[6]),
        .bcd7(Data[7])
    );

    Divider100MHzTo1kHzAnd1Hz uut_divider(
        .clk_100MHz(clk_100mhz),
        .clk_1kHz(clk_1000hz),
        .clk_1Hz(clk_1hz)
    );

    always @(posedge clk_1hz) begin
        if(rst==0)
            stime <= 0;
        else if (isfinish==1)
            stime <= stime;
        else
            stime <= stime + 1;
    end

    always @(posedge clk_1000hz) begin
        if (cnt == 4'd7)
            cnt <= 0;
        else
            cnt <= cnt + 1;

        shift <= 8'b1111_1111;
```

```
            shift[cnt] <= 0;

            case (Data[cnt])
                4'b0000: oData <= 7'b1000000;
                4'b0001: oData <= 7'b1111001;
                4'b0010: oData <= 7'b0100100;
                4'b0011: oData <= 7'b0110000;
                4'b0100: oData <= 7'b0011001;
                4'b0101: oData <= 7'b0010010;
                4'b0110: oData <= 7'b0000010;
                4'b0111: oData <= 7'b1111000;
                4'b1000: oData <= 7'b0000000;
                4'b1001: oData <= 7'b0010000;
                default: oData <= 7'b1111111;
            endcase
        end
endmodule

initial
begin
    key_data=1;
    forever #2   key_data=~key_data;
end

endmodule

module keyboard//将键盘键码转化为ASCII 码
(
input                   clk_in,             //系统时钟
input                   rst,                //系统复位，低有效
input                   key_clk,            //PS2 键盘时钟输入
input                   key_data,           //PS2 键盘数据输入
output  reg             key_state,          //键盘的按下状态，按下置1，松开置0
output  reg     [7:0]   key_ascii           //按键键值对应的ASCII 编码
);

reg     key_clk_r0 = 1'b1,key_clk_r1 = 1'b1;
reg     key_data_r0 = 1'b1,key_data_r1 = 1'b1;
//对键盘时钟数据信号进行延时锁存
always @ (posedge clk_in or negedge rst)
begin
    if(!rst)
    begin
        key_clk_r0 <= 1'b1;
```

```
            key_clk_r1 <= 1'b1;
            key_data_r0 <= 1'b1;
            key_data_r1 <= 1'b1;
        end
        else
        begin
            key_clk_r0 <= key_clk;
            key_clk_r1 <= key_clk_r0;
            key_data_r0 <= key_data;
            key_data_r1 <= key_data_r0;
        end
    end

    //键盘时钟信号下降沿检测
    wire     key_clk_neg = key_clk_r1 & (~key_clk_r0);
    reg              [3:0]    cnt;
    reg              [7:0]    temp_data;
    //根据键盘的时钟信号的下降沿读取数据
    always @ (posedge clk_in or negedge rst)
    begin
        if(!rst)
        begin
            cnt <= 4'd0;
            temp_data <= 8'd0;
        end
        else if(key_clk_neg)
        begin
            if(cnt >= 4'd10) cnt <= 4'd0;
            else cnt <= cnt + 1'b1;
            case (cnt)
                4'd0: ; //起始位
                4'd1: temp_data[0] <= key_data_r1;   //数据位 bit0
                4'd2: temp_data[1] <= key_data_r1;   //数据位 bit1
                4'd3: temp_data[2] <= key_data_r1;   //数据位 bit2
                4'd4: temp_data[3] <= key_data_r1;   //数据位 bit3
                4'd5: temp_data[4] <= key_data_r1;   //数据位 bit4
                4'd6: temp_data[5] <= key_data_r1;   //数据位 bit5
                4'd7: temp_data[6] <= key_data_r1;   //数据位 bit6
                4'd8: temp_data[7] <= key_data_r1;   //数据位 bit7
                4'd9: ; //校验位
                4'd10:; //结束位
                default: ;
            endcase
        end
```

```
end

reg key_break = 1'b0;
reg [7:0]key_byte = 1'b0;
//根据通码和断码判定按键的当前是按下还是松开
always @ (posedge clk_in or negedge rst)
begin
    if(!rst)
        begin
            key_break <= 1'b0;
            key_state <= 1'b0;
            key_byte <= 1'b0;
        end
    else if(cnt==4'd10 && key_clk_neg)
        begin
            if(temp_data == 8'hf0)
                key_break <= 1'b1;  //收到断码（8'hf0）表示按键松开，下一个数据为断码，设置断码标示为1
            else if(!key_break)
                begin    //当断码标示为0 时，表示当前数据为按下数据，输出键值并设置按下标示为1
                    key_state <= 1'b1;
                    key_byte <= temp_data;
                end
            else
                begin    //当断码标示为1 时，标示当前数据为松开数据，断码标示和按下标示都清0
                    key_state <= 1'b0;
                    key_break <= 1'b0;
                    key_byte<=0;
                end
        end
end

//将键盘返回的有效键值转换为按键字母对应的ASCII 码
always @ (key_byte) begin
    case (key_byte)     //把ps2 返回的转化成ascii 码
        8'h75: key_ascii <= 8'd1;//上箭头
        8'h72: key_ascii <= 8'd2;//下箭头
        8'h6b: key_ascii <= 8'd3;//左箭头
        8'h74: key_ascii <= 8'd4;//右箭头
        8'h29: key_ascii <= 8'd13;    //回车
        8'h1D: key_ascii <= 8'h57;    // W
        8'h1C: key_ascii <= 8'h41;    // A
```

```
        8'h1B: key_ascii <= 8'h53;    // S
        8'h23: key_ascii <= 8'h44;    // D
        default: key_ascii=0;         //nothing
    endcase
end

endmodule

//本模块的主要用处是根据上下左右 wasd 键控制 led 灯的亮灭，这样就可以根据 led 的反馈肉眼看出自己的操作是否正确
module led(
    input key_state       ,//键盘按键状态
    input [8:0]keys       ,//按下键盘的 ASCII 值
    output reg [4:0] led //控制哪个 led 灯亮起,分别代表前后左右以及复位（）
);
    always@(key_state)
        begin
            led<=5'b0;//每次启动的时候重置 led 灯信号，这样保证了每次只有一个 led 灯会亮起来
            if (keys== 8'd1||keys== 8'h57) begin
            led<=5'b00001;//前
            end
            else if (keys== 8'd2||keys== 8'h53) begin
            led<=5'b00010;//后
            end
            else if (keys== 8'd3||keys== 8'h41) begin
            led<=5'b00100;//左
            end
            else if (keys== 8'd4||keys== 8'h44) begin
            led<=5'b01000;//右
            end
            else if (keys== 8'd13||keys== 8'h58) begin
            led<=5'b10000;//复位
            end
        end
endmodule

module migong_top(
    input clk_100,   //100mhz
    input key_clk,   //键盘时钟
    input key_data, //键盘输入数据
    input rst,//低电平复位
    output wire hsync,
    output wire vsync,
```

```
    output wire [4:0] led,
    output wire [11:0]vga_rgb,
    output wire [7:0] shift,
    output wire [6:0] oData
    );
    //存储迷宫地图16×16
reg [0:255] map =
    {
        16'b1001_1111_1111_1111,
        16'b1000_0000_0001_0001,
        16'b1111_1111_0001_1111,
        16'b1001_0001_1000_0001,
        16'b1101_0101_1001_0111,
        16'b1001_0100_1001_0001,
        16'b1000_0100_0010_0001,
        16'b1001_1111_0011_0101,
        16'b1011_0101_0001_0101,
        16'b1101_0101_0001_0111,
        16'b1000_0101_0010_0001,
        16'b1000_0101_0011_1111,
        16'b1000_0101_0010_0101,
        16'b1011_1101_1000_0101,
        16'b1000_0000_0010_0001,
        16'b1111_1111_1111_1001
    };

    wire [8:0] keys;
    wire key_state;
    wire [9:0]px;
    wire [9:0]py;
    wire is_finish;

    vga_caculate    u_vga_caculate(
        .map(map),
        .key_clk(key_clk),
        .keys(keys),
        .px(px),
        .py(py),
        .is_finish(is_finish)
    );

//VGA 显示
    vga_top u_vga(
        .map(map),
```

```
            .sys_clk(clk_100),
            .sys_rst_n(rst),
            .px(px),
            .py(py),
            .is_finish(is_finish),
            .hsync(hsync),
            .vsync(vsync),
            .vga_rgb(vga_rgb)
        );

    //键盘
        keyboard uut_keyboard(
            .clk_in(clk_100),
            .rst(rst),
            .key_clk(key_clk),
            .key_data(key_data),
            .key_state(key_state),
            .key_ascii(keys)
        );

    //led
        led uut_led(
            .key_state(key_state),
            .keys(keys),
            .led(led)
        );
     display_stime u_display_stime(
        .rst(rst),
        .isfinish(is_finish),
        .clk_100mhz(clk_100),
        .shift(shift),
        .oData(oData)
    );
    endmodule
    module vga_caculate(
        input wire[0:255]map,//存储迷宫地图
        input wire key_clk,//键盘时钟
        input wire[8:0]keys,//键盘输入信息
        output [9:0]px,//横坐标
        output [9:0]py,//纵坐标
        output is_finish//判断是否结束的标志
        );
        reg [9:0]px1=50;
        reg [9:0]py1=10;
```

```
    reg is_finish1=0;
    always@(negedge key_clk)
        begin
            begin
                case(keys)
                8'd1:
                    if(map[px1/40+(py1-1'b1)/30*16]==0)
                        py1=py1-1'b1;
                8'd2:
                    if(map[px1/40+(py1+1'b1)/30*16]==0)
                        py1=py1+1'b1;
                8'd3:
                    if(map[(px1-1'b1)/40+py1/30*16]==0)
                        px1=px1-1'b1;
                8'd4:
                    if(map[(px1+1'b1)/40+py1/30*16]==0)
                        px1=px1+1'b1;
                8'd13:
                    begin
                        px1=50;
                        py1=10;
                        is_finish1=0;
                    end
                8'h57:
                    if(map[px1/40+(py1-1'b1)/30*16]==0)
                        py1=py1-1'b1;
                8'h53:
                    if(map[px1/40+(py1+1'b1)/30*16]==0)
                        py1=py1+1'b1;
                8'h41:
                    if(map[(px1-1'b1)/40+py1/30*16]==0)
                        px1=px1-1'b1;
                8'h44:
                    if(map[(px1+1'b1)/40+py1/30*16]==0)
                        px1=px1+1'b1;
                endcase
            end
            if(px1/40>=13&&px1/40<=15&&py1/30>=15)
                is_finish1=1;
        end
    assign px=px1;
    assign py=py1;
    assign is_finish=is_finish1;
endmodule
```

```
module   vga_control
(
    input    wire            vga_clk      ,//25MHZ 时钟
    input    wire            sys_rst_n    ,//复位信号
    input    wire    [11:0]  pix_data     ,//当前像素的 rgb 数据
    output   wire    [9:0]   pix_x        ,//当前像素横坐标
    output   wire    [9:0]   pix_y        ,//当前像素纵坐标
    output   wire            hsync        ,//行同步信号
    output   wire            vsync        ,//场同步信号
    output   wire    [11:0]  vga_rgb       //输出三色 rgb
);
//总分辨率为 640×480
//hsync800
parameter    H_SYNC    = 10'd96 ,//同步脉冲
             H_BACK    = 10'd40 ,//显示后沿
             H_LEFT    = 10'd8  ,//左边框
             H_VALID   = 10'd640,//宽度设置 640 个像素
             H_RIGHT   = 10'd8  ,//右边框
             H_FRONT   = 10'd8  ,//显示前沿
             H_TOTAL   = 10'd800;//帧长
//vsync525
parameter    V_SYNC    = 10'd2  ,//同步脉冲
             V_BACK    = 10'd25 ,//显示后沿
             V_TOP     = 10'd8  ,//上边框
             V_VALID   = 10'd480,//高度设置为 480 个像素
             V_BOTTOM = 10'd8   ,//下边框
             V_FRONT   = 10'd2  ,//显示前沿
             V_TOTAL   = 10'd525;//帧长

reg [9:0] h_count; //  水平扫描计数器
reg [9:0] v_count; //  垂直扫描计数器

wire             pix_data_req;
wire             rgb_valid    ;

//cnt_h
always @(posedge vga_clk or negedge sys_rst_n) begin
        if (~sys_rst_n)
            h_count <= 10'd0;
        else if (h_count == H_TOTAL - 1)
            h_count <= 10'd0;
        else
            h_count <= h_count + 1'b1;
```

```
    end

    // 垂直计数器逻辑
always @(posedge vga_clk or negedge sys_rst_n) begin
        if (~sys_rst_n)
            v_count <= 10'd0;
        else if ((v_count == V_TOTAL - 1) && (h_count == H_TOTAL - 1))
            v_count <= 10'd0;
        else if (h_count == H_TOTAL - 1)
            v_count <= v_count + 1'b1;
        else
            v_count <= v_count;
end

//rgb_valid
//若 cnt_h 和 cnt_v 都在显示范围之内则 rgb_valid 为 1 否则为 0
assign   rgb_valid = (h_count >= (H_SYNC + H_BACK + H_LEFT)) && (h_count < (H_SYNC
+ H_BACK + H_LEFT + H_VALID)) &&
                    (v_count >= (V_SYNC + V_BACK + V_TOP)) && (v_count < (V_SYNC +
V_BACK + V_TOP + V_VALID))
                    ? 1'b1 : 1'b0;

//pix_data_req
//若 cnt_h 和 cnt_v 都在显示范围之内则 pix_data_req 为 1 否则为 0
assign pix_data_req = (h_count >= (H_SYNC + H_BACK + H_LEFT - 1)) && (h_count <
(H_SYNC + H_BACK + H_LEFT + H_VALID - 1)) &&
                      (v_count >= (V_SYNC + V_BACK + V_TOP)) && (v_count < (V_SYNC +
V_BACK + V_TOP + V_VALID))
                    ? 1'b1 : 1'b0;

//通过 cnt_h 和 cnt_v 计算得到 pix_x 和 pix_y 的准确位置
assign pix_x = (pix_data_req == 1'b1) ? (h_count - (H_SYNC + H_BACK + H_LEFT - 1'b1)) :
10'h3ff;
assign pix_y = (pix_data_req == 1'b1) ? (v_count - (V_SYNC + V_BACK + V_TOP)) : 10'h3ff;
//返回行同步和场同步信号
assign hsync = (h_count <= H_SYNC - 1'b1) ? 1'b1 : 1'b0;
assign vsync = (v_count <= V_SYNC - 1'b1) ? 1'b1 : 1'b0;
//返回三色 rgb 信号
assign vga_rgb = (rgb_valid == 1'b1) ? pix_data : 12'h000;

endmodule
module   vga_data
(
    input     wire [0:255]map,//存储地图信息
```

```
    input    wire vga_clk,//65MHZ 时钟
    input    wire sys_rst_n,//复位信号
    input    wire [9:0] pix_x,//当前像素横坐标
    input    wire [9:0] pix_y,//当前像素纵坐标
    input    wire [9:0] px,//红点横坐标
    input    wire [9:0] py,//红点纵坐标
    input    wire is_finish,//判断是否结束的标志
    output   reg [11:0] pix_data//当前像素点的三色 rgb 信息
);
//设定需要颜色的 RGB 编码
parameter    RED     = 12'hF00,
             YELLOW = 12'hFF0,
             CYAN    = 12'h0FF,//青色
             BLUE    = 12'h00F,
             PURPLE = 12'hF0F,
             BLACK   = 12'h000,
             WHITE   = 12'hFFF;

//存储字模："逃离成功！"
reg [0:1279]congrats_msg={
    8'b00000000, 8'b10100000, 8'b00000010, 8'b00000000, 8'b00000000, 8'b01010000,
8'b00000000, 8'b01000000, 8'b00000000, 8'b00000000,
    8'b00100000, 8'b10100000, 8'b00000001, 8'b00000000, 8'b00000000, 8'b01001000,
8'b00000000, 8'b01000000, 8'b00010000, 8'b00000000,
    8'b00010100, 8'b10100100, 8'b11111111, 8'b11111110, 8'b00000000, 8'b01000000,
8'b00000000, 8'b01000000, 8'b00010000, 8'b00000000,
    8'b00010010, 8'b10101000, 8'b00000000, 8'b00000000, 8'b00111111, 8'b11111110,
8'b11111110, 8'b01000000, 8'b00010000, 8'b00000000,
    8'b00000001, 8'b10110000, 8'b00010100, 8'b01010000, 8'b00100000, 8'b01000000,
8'b00010001, 8'b11111100, 8'b00010000, 8'b00000000,
    8'b00000000, 8'b10100000, 8'b00010011, 8'b10010000, 8'b00100000, 8'b01000000,
8'b00010000, 8'b01000100, 8'b00010000, 8'b00000000,
    8'b11110001, 8'b10110000, 8'b00010100, 8'b01010000, 8'b00100000, 8'b01000100,
8'b00010000, 8'b01000100, 8'b00010000, 8'b00000000,
    8'b00010010, 8'b10101000, 8'b00011111, 8'b11110000, 8'b00111110, 8'b01000100,
8'b00010000, 8'b01000100, 8'b00010000, 8'b00000000,
    8'b00010100, 8'b10100100, 8'b00000001, 8'b00000000, 8'b00100010, 8'b01000100,
8'b00010000, 8'b01000100, 8'b00010000, 8'b00000000,
    8'b00010001, 8'b00100000, 8'b01111111, 8'b11111100, 8'b00100010, 8'b00101000,
8'b00010000, 8'b10000100, 8'b00010000, 8'b00000000,
    8'b00010001, 8'b00100100, 8'b01000010, 8'b00000100, 8'b00100010, 8'b00101000,
8'b00010000, 8'b10000100, 8'b00000000, 8'b00000000,
    8'b00010010, 8'b00100100, 8'b01000100, 8'b01000100, 8'b00100010, 8'b00010010,
8'b00011110, 8'b10000100, 8'b00000000, 8'b00000000,
```

```
        8'b00010100, 8'b00011100, 8'b01001111, 8'b11100100, 8'b00101010, 8'b00110010,
8'b11110001, 8'b00000100, 8'b00010000, 8'b00000000,
        8'b00101000, 8'b00000000, 8'b01000100, 8'b00100100, 8'b01000100, 8'b01001010,
8'b01000001, 8'b00000100, 8'b00010000, 8'b00000000,
        8'b01000111, 8'b11111110, 8'b01000000, 8'b00010100, 8'b01000000, 8'b10000110,
8'b00000010, 8'b00101000, 8'b00000000, 8'b00000000,
        8'b00000000, 8'b00000000, 8'b01000000, 8'b00001000, 8'b10000001, 8'b00000010,
8'b00000100, 8'b00010000, 8'b00000000, 8'b00000000
};

always@(posedge vga_clk )
    begin
        if(is_finish==0)
            if(sys_rst_n == 1'b0)
                pix_data <= PURPLE;
            else if(pix_x>=px-2&&pix_x<=px+2&&pix_y>=py-2&&pix_y<=py+2)
                pix_data<=PURPLE;
            else if(map[pix_x/40+pix_y/30*16]==1)
                pix_data<=YELLOW;
            else
                pix_data<=WHITE;
        else
             if(pix_x>=50&&pix_x<450&&pix_y>=100&&pix_y<180&&congrats_msg[pix_x/5-
10+(pix_y/5-20)*80]==1)
              pix_data <= PURPLE;
            else
                pix_data<=WHITE;
    end
endmodule
module   vga_top
(
    input    wire     [0:255] map          ,//存储地图信息
    input    wire             sys_clk      ,//系统时钟
    input    wire             sys_rst_n    ,//复位信号
    input    wire     [9:0]    px           ,//红点横坐□?
    input    wire     [9:0]    py           ,//红点纵坐□?
    input    wire             is_finish    ,//是否到达终点标记
    output   wire             hsync        ,//行同步信□?
    output   wire             vsync        ,//场同步信□?
    output   wire     [11:0]   vga_rgb       //输出三色 rgb
);

wire              vga_clk      ;
wire              locked       ;
```

```
wire              rst_n         ;
wire     [9:0]    pix_x         ;
wire     [9:0]    pix_y         ;
wire     [11:0]   pix_data      ;

assign   rst_n = (sys_rst_n && locked);
//分频IP□?
clk_wiz_0 clk_gen_inst
(
    .clk_in1     (sys_clk),       // 输入系统时钟 100mhz
    .reset       (~sys_rst_n),    // 输入复位信号
    .clk_out1    (vga_clk),       // 得到降频后的时钟信号 25.175mhz
    .locked      (locked)         // 得到 locked 信号
);
//vga 信号控制模块
vga_control     u_vga_control
(
    .vga_clk      (vga_clk   ),
    .sys_rst_n    (rst_n     ),
    .pix_data     (pix_data ),
    .pix_x        (pix_x     ),
    .pix_y        (pix_y     ),
    .hsync        (hsync     ),
    .vsync        (vsync     ),
    .vga_rgb      (vga_rgb   )
);
//rgb 颜色选择模块
vga_data     u_vga_data
(
    .map          (map       ),
    .vga_clk      (vga_clk   ),
    .sys_rst_n    (rst_n     ),
    .pix_x        (pix_x     ),
    .pix_y        (pix_y     ),
    .px           (px        ),
    .py           (py        ),
    .is_finish    (is_finish),
    .pix_data     (pix_data )
);

endmodule
```

测试文件:

```
`timescale 1ns / 1ps
```

```
module vga_top_tb;
parameter PERIOD   = 2;
reg    [0:255]   map                          = 0 ;
reg    sys_clk                                = 0 ;
reg    sys_rst_n                              = 0 ;
reg    [9:0]   px                             = 0 ;
reg    [9:0]   py                             = 0 ;
reg    is_finish                              = 0 ;
wire   hsync                                  ;
wire   vsync                                  ;
wire   [11:0]   vga_rgb                       ;

initial
begin
    forever #(PERIOD/2)   sys_clk=~sys_clk;
end

initial
begin
    #(PERIOD*2) sys_rst_n   =   1;
end

vga_top   u_vga_top (
    .map                    ( map              ),
    .sys_clk                ( sys_clk          ),
    .sys_rst_n              ( sys_rst_n        ),
    .px                     ( px               ),
    .py                     ( py               ),
    .is_finish              ( is_finish        ),
    .hsync                  ( hsync            ),
    .vsync                  ( vsync            ),
    .vga_rgb                ( vga_rgb          )
);

endmodule

`timescale   10ns / 1ps
module keyboard_tb;
reg    clk_in                                 = 0 ;
reg    rst                                    = 0 ;
reg    key_clk                                = 0 ;
reg    key_data                               = 0 ;
wire   key_state                              ;
wire   [7:0]   key_ascii                      ;
```

```
keyboard   u_keyboard (
    .clk_in                 ( clk_in            ),
    .rst                    ( rst               ),
    .key_clk                ( key_clk           ),
    .key_data               ( key_data          ),
    .key_state              ( key_state         ),
    .key_ascii              ( key_ascii         )
);
initial
begin
    clk_in=0;
    forever #1 clk_in=~clk_in;
end

initial
begin
    key_clk=1;
    forever #10   key_clk=~key_clk;
end

initial
begin
    rst=0;
    #5 rst=1;
end

initial
begin
    key_data=1;
    forever #2   key_data=~key_data;
end

endmodule
```