**北京科技大学 计算机与通信工程学院**

**实 验 报 告**

**实验名称**：\_\_\_\_\_\_\_\_\_\_状态机实验\_\_\_\_\_\_\_\_\_\_\_

**学生姓名**：\_\_\_\_\_\_\_\_\_\_唐誉源\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**专 业**：\_\_\_\_\_\_\_\_\_计算机技术与科学\_\_\_\_\_\_\_\_\_

**班 级**：\_\_\_\_\_\_\_\_\_\_\_\_计1503\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**学 号**：\_\_\_\_\_\_\_\_\_\_\_\_41503302\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**指导教师**：\_\_\_\_\_\_\_\_\_\_何杰\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**实验成绩**：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**实验地点**：\_\_\_\_\_\_\_\_\_\_机电楼304\_\_\_\_\_\_\_\_\_\_\_\_\_

**实验时间**：\_\_\_2017\_\_\_\_\_\_年\_\_4\_月\_\_3\_\_\_日

**一、实验目的与实验要求**

**1、实验目的**

综合运用本课程所学习的知识，设计并实现复杂的数字系统。

**2、实验要求**

1. 在实验报告中提交状态图或者算法流程状态图，系统级模块图、设计代码、仿真程序、仿真结果截图、实测验证结果照片。
2. 制作复杂数字系统设计报告**PPT**，验收过程除了演示自己的作品外，还需要通过**PPT**讲解数字系统设计。**PPT**讲解时间**2**分钟。
3. 提交实验报告、PPT和每个实验的完整工程文件。

**二、实验设备（环境）及要求**

实验室提供Windows 7操作系统下的Xilinx开发平台vivado2015.4环境，以及EGO开发板。

**三、实验内容与步骤**（类似于“**解决问题**”）

**1.**实验内容

从**实验5.1、实验5.2、实验5.3中选择其中一个完成即可**，本实验评分依据系统复杂度、完成程度、系统展示效果等进行综合评价。

**实验5.1** 自主设计综合实验（要求一~六是必须完成；满足要求一~六且加分项达到3项或者以上，则系统复杂度为满分）

* 要求一：自己确定实验内容；
* 要求二：实验内容可以通过LED或者VGA显示器(参考本文档4.1 和 第21页~26页 5.5~5.7)来展示；
* 要求三：除了用于展示实验结果的硬件外，还至少要用到EGO1实验板上的至少2种硬件。
* 要求四：划分数据通路和控制电路
* 要求五：控制电路用状态机实现，至少包含4种状态。
* 要求六：至少用到加法器、乘法器、除法器、计数器、分频器、移位寄存器等基本中的4种（可以利用实验室一~四的代码，以及实验5.2 中的）。
* **加分项**：使用流水线、使用3个以上状态机、单个状态机状态数量超过10个、使用VGA显示器、使用SRAM 超过100Kbyte、 使用EGO1上除了按键/LED/数码管/SRAM以外的其他硬件（蓝牙、UART通信可以参考“实验5.1 EGO1 蓝牙、UART通信.zip”）、以及其他有难度的设计。

**实验5.**2 俄罗斯方块（完成基础版本且增加额外功能达到2项，则系统复杂度得分为满分）

* 根据“4. 俄罗斯方块实验说明和5.俄罗斯方块实验步骤”完成俄罗斯方块游戏的基础版本开发；
* 在基础版本上完善设计，增加额外功能：例如上述实验中并未涉及分数统计，请在原设计的基础上添加代码完成分数统计的工作，并将分数实时显示在数码管上，还可将分数保存到存储器当中，进行排序，显示纪录等。

**实验5.3 自主游戏设计实验**（系统复杂度为满分）

* 游戏题目自拟（俄罗斯方块除外），必须通过VGA显示。

2.主要步骤

游戏规则：

对于任意细胞: 每个细胞有两种状态(存活或死亡)，且与以自身为中心的周围八格细胞产生互动。

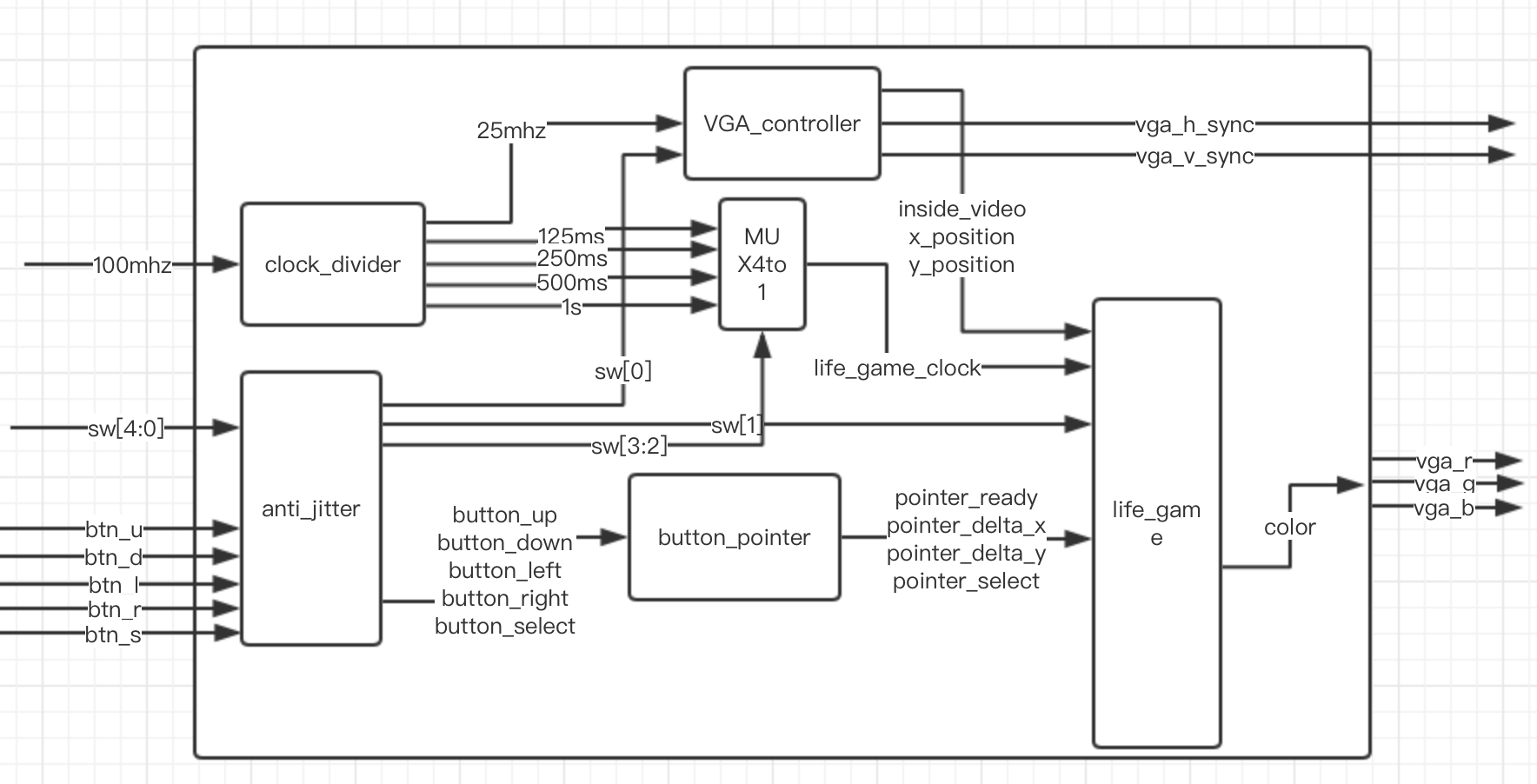
1 当前细胞为存活状态时，当周围低于2个( 包含2个)存活细胞时，该细胞变成死亡状态。(模拟生命数稀少)

2 当前细胞为存活状态时，当周围有2个或3个存活细胞时，该细胞保持原样。

3 当前细胞为存活状态时，当周围有3个以上的存活细胞时，该细胞变成死亡状态。(模拟生命数过多)

4 当前细胞为死亡状态时，当周围有3个存活细胞时，该细胞变成存活状态。 (模拟繁殖)

可以把最初的细胞结构定义为种子，当所有在中的细胞同时被以上规则处理后, 可以 得到第一代细胞图。按规则继续处理当前的细胞图，可以得到下一代的细胞图，周而复始。

 系统级模块设计图

TOP模块代码：

module top(

input clock,

input [7:0] sw,

input btn\_l, btn\_r, btn\_u, btn\_d, btn\_s,

output reg [7:0] led = 0,

output vga\_h\_sync,

output vga\_v\_sync,

output [3:0] vga\_r,

output [3:0] vga\_g,

output [3:0] vga\_b

);

wire [7:0] switch;

wire button\_left, button\_right, button\_up, button\_down, button\_select;

wire clock\_50mhz;

wire pointer\_ready;

wire [8:0] pointer\_delta\_x;

wire [8:0] pointer\_delta\_y;

wire clock\_25mhz;

wire inside\_video;

wire [9:0] x\_position;

wire [8:0] y\_position;

wire [11:0] color;

wire clock\_125ms;

wire clock\_250ms;

wire clock\_500ms;

wire clock\_1s;

wire clock\_life\_game;

anti\_jitter #(8) switch\_anti\_jitter(clock, sw[7:0], switch[7:0]);

anti\_jitter #(5) button\_anti\_jitter(clock, {btn\_l, btn\_r, btn\_u, btn\_d, btn\_s}, {button\_left, button\_right, button\_up, button\_down, button\_select});

clock\_divider #(1) clock\_divider\_to\_50mhz(clock, clock\_50mhz);

clock\_divider #(1) clock\_divider\_to\_25mhz(clock\_50mhz, clock\_25mhz);

vga\_controller vga\_controller(clock\_25mhz, switch[0], vga\_h\_sync, vga\_v\_sync, inside\_video, x\_position, y\_position);

// button\_pointer #(25) button\_pointer (button\_left, button\_right, button\_up, button\_down, button\_select, pointer\_ready, pointer\_delta\_x, pointer\_delta\_y, pointer\_select);

button\_pointer #(25) button\_pointer (button\_up, button\_down, button\_left, button\_right, button\_select, pointer\_ready, pointer\_delta\_x, pointer\_delta\_y, pointer\_select);

clock\_divider\_to\_125ms clock\_divider\_to\_125ms(clock, clock\_125ms);

clock\_divider #(1) clock\_divider\_to\_250ms(clock\_125ms, clock\_250ms);

clock\_divider #(1) clock\_divider\_to\_500ms(clock\_250ms, clock\_500ms);

clock\_divider #(1) clock\_divider\_to\_1s(clock\_500ms, clock\_1s);

multiplexer\_4\_1 clock\_multiplexer({clock\_125ms, clock\_250ms, clock\_500ms, clock\_1s}, switch[6:5], clock\_life\_game);

life\_game #(25) life\_game(clock\_life\_game, switch[7], switch[4],pointer\_ready, pointer\_delta\_x, pointer\_delta\_y, pointer\_select, x\_position, y\_position, inside\_video, color);

assign vga\_r = color[11:8];

assign vga\_g = color[7:4];

assign vga\_b = color[3:0];

always @(posedge clock\_125ms) begin

if (sw[0]) led[0] = 1;

else if (!sw[0]) led[0] = 0;

if (sw[1]) led[1] = 1;

else if (!sw[1]) led[1] = 0;

if (sw[2]) led[2] = 1;

else if (!sw[2]) led[2] = 0;

if (sw[3]) led[3] = 1;

else if (!sw[3]) led[3] = 0;

if (sw[4]) led[4] = 1;

else if (!sw[4]) led[4] = 0;

if (sw[5]) led[5] = 1;

else if (!sw[5]) led[5] = 0;

if (sw[6]) led[6] = 1;

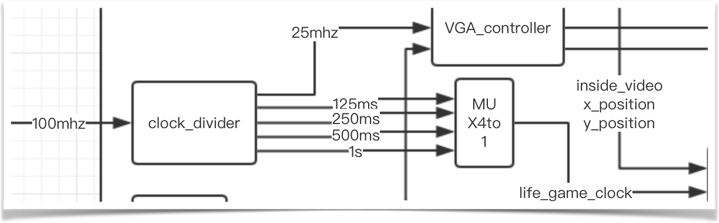
else if (!sw[6]) led[6] = 0;

if (sw[7]) led[7] = 1;

else if (!sw[7]) led[7] = 0;

end

endmodule



时钟分频模块

//将100mhz的系统时钟频率分频为25mhz输入VGA控制模块。

//将100mhz的系统时钟频率分频为周期分别为125ms,250ms,500ms以及1s的时钟频率输入给四选一多路选通器，输出不同的频率，作为输入给life\_game，可以控制细胞演进速度。

module clock\_divider(

input in,

output out

);

parameter COUNTER\_WIDTH = 1;

reg [COUNTER\_WIDTH - 1:0] counter = 0;

always @(posedge in) begin

counter <= counter + 1'b1;

end

assign out = counter[COUNTER\_WIDTH - 1];

endmodule

module clock\_divider\_to\_125ms(

input clock\_100mhz,

output reg clock\_125ms = 0

);

parameter COUNTER\_MAX = 6250000;

reg [25:0] counter = 0;

always @(posedge clock\_100mhz) begin

if (counter < COUNTER\_MAX - 1) begin

counter <= counter + 1'b1;

end else begin

counter <= 0;

clock\_125ms <= ~clock\_125ms;

end

end

endmodule

`timescale 1ns / 1ps

四选一多路选通器：

module multiplexer\_4\_1(

input [3:0] in,

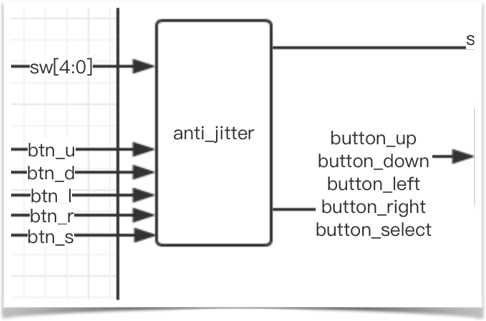
input [1:0] select,

output out

);

assign out = in[select];

endmodule



输入消抖模块

输入消抖模块代码：

parameter HOLD = 100000;//计时到32个时间单位

reg [WIDTH - 1:0] last\_seen = 0;

always @(posedge clock) begin

if (last\_seen != in) begin

counter <= 0;//如果输入改变则重新计时

end else if (counter < HOLD) begin

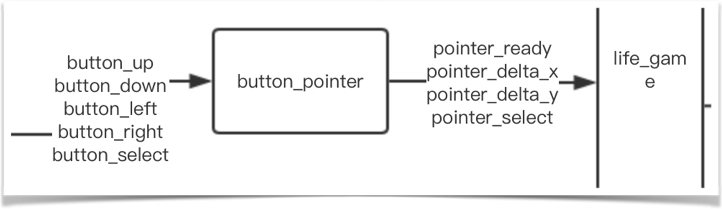
counter <= counter + 1;

end else begin //输入一个按键，其后的32个时间单位内不改变，则有效

out <= last\_seen; end

last\_seen <= in;//将每一时刻的输入记录

end



按键处理模块

//将经过消陡处理的按键信号进行处理，输入到life\_game中，方便life\_game中判断//输入信号，产生相应响应

//输入改变时，pointer\_ready有效

//button\_select有效时，pointer\_select有效

//pointer\_delta\_x和pointer\_delta\_y区别了上下左右

`timescale 1ns / 1ps

module button\_pointer(

input button\_left,

input button\_right,

input button\_up,

input button\_down,

input button\_select,

output pointer\_ready,

output [8:0] pointer\_delta\_x,

output [8:0] pointer\_delta\_y,

output pointer\_select

);

parameter DELTA\_UNIT = 7'd20;

//当存在按键输入时pointer\_ready有效

assign pointer\_ready = button\_left || button\_right || button\_up || button\_down || button\_select;

//区别左右移动，方便life\_game模块通过加减运算改变方格位置

assign pointer\_delta\_x[8] = button\_left;

assign pointer\_delta\_x[7:0] = button\_left || button\_right ? DELTA\_UNIT : 7'd0;

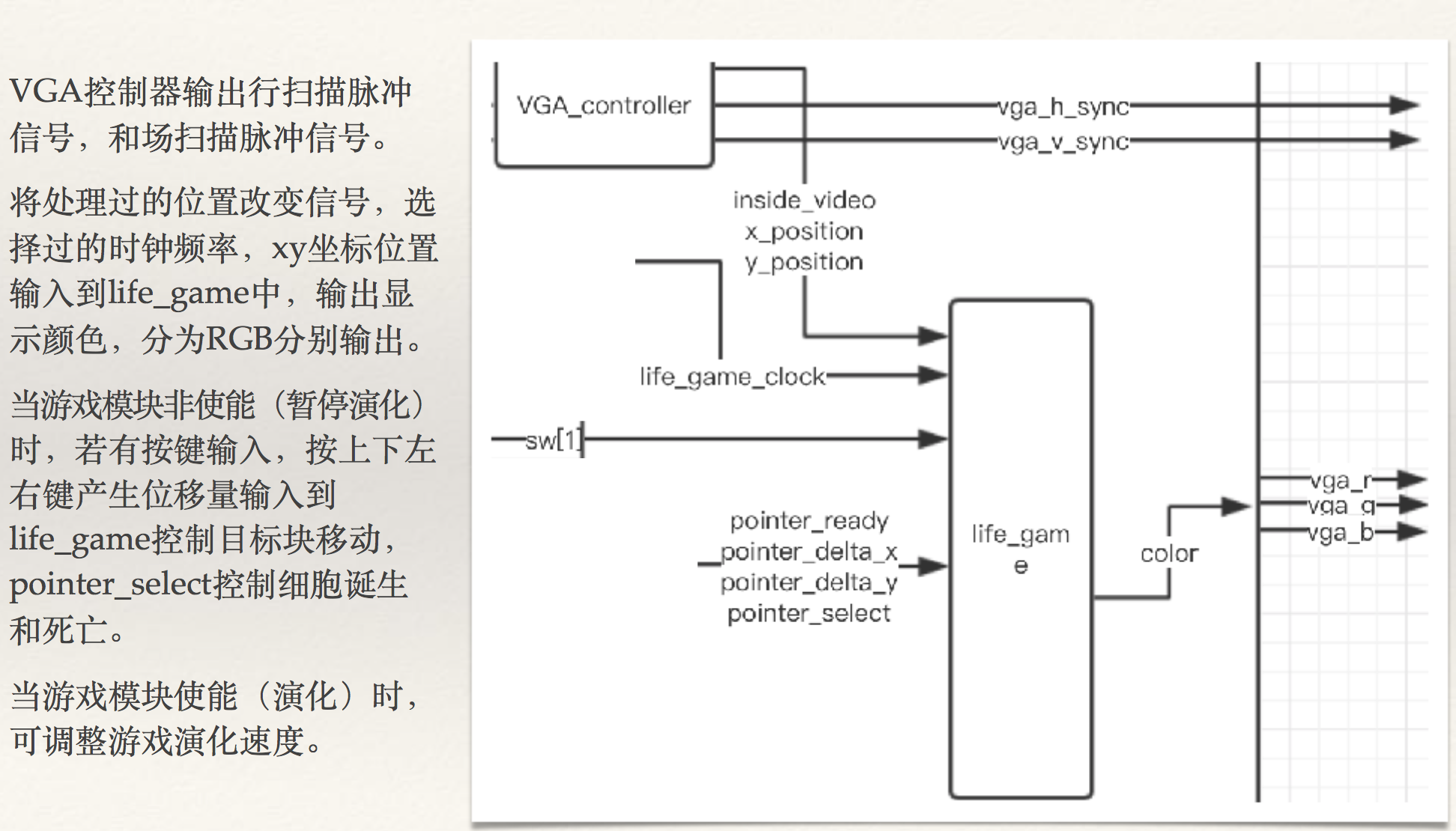
//区别上下移动，方便life\_game模块通过加减运算改变方格位置

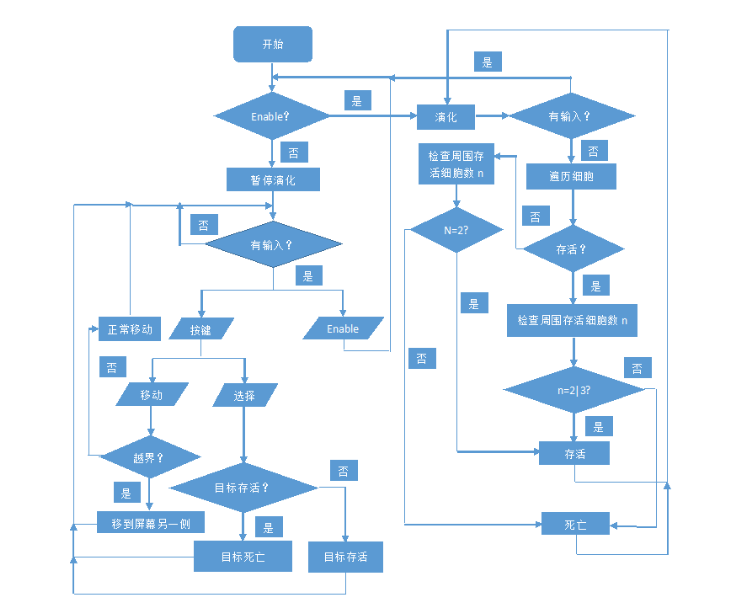
assign pointer\_delta\_y[8] = button\_up;

assign pointer\_delta\_y[7:0] = button\_up || button\_down ? DELTA\_UNIT : 7'd0;

assign pointer\_select = button\_select;

endmodule

游戏控制模块



ASM算法状态图

游戏模块代码：

module life\_game(

input clock,

input enable, //是否活动

input circlize,

input pointer\_ready, //按下键即为有效

input [8:0] pointer\_delta\_x, //目标各自deltax值

input [8:0] pointer\_delta\_y,

input pointer\_select,

input [9:0] x\_position, //显示区内坐标 640×480

input [8:0] y\_position,

input inside\_video, //显示区内

output reg [11:0] color //8位rgb

);

//以下是UI端口

parameter BLOCK\_SIZE = 20; //格子大小

parameter BLOCK\_COUNT\_X = 640 / BLOCK\_SIZE; //行格子数

parameter BLOCK\_COUNT\_Y = 480 / BLOCK\_SIZE; //列格子数

parameter COLOR\_POINTER = 12'b0000\_1100\_1000; //颜色

parameter COLOR\_EMPTY = 12'b1111\_1111\_1111;

parameter COLOR\_BLACK = 12'b0000\_0000\_0000;

reg [9:0] pointer\_x = 320; //初始目标坐标

reg [8:0] pointer\_y = 240;

wire [4:0] pointer\_x\_index; //目标索引

wire [4:0] pointer\_y\_index;

wire pointer\_select\_on [0:BLOCK\_COUNT\_X - 1] [0:BLOCK\_COUNT\_Y - 1];

reg map\_0 [0:BLOCK\_COUNT\_X - 1] [0:BLOCK\_COUNT\_Y - 1];

reg map\_1 [0:BLOCK\_COUNT\_X - 1] [0:BLOCK\_COUNT\_Y - 1];

reg map\_index = 0;

wire [3:0] neighbor\_count\_0 [0:BLOCK\_COUNT\_X - 1] [0:BLOCK\_COUNT\_Y - 1]; //周围细胞数量

wire [3:0] neighbor\_count\_1 [0:BLOCK\_COUNT\_X - 1] [0:BLOCK\_COUNT\_Y - 1];

wire [4:0] x\_index;

wire [4:0] y\_index;

wire block\_current;

wire [11:0] color\_life;

wire [11:0] color\_empty;

wire [11:0] color\_block;

wire [11:0] color\_block\_circlized;

initial begin

map\_0[1][0] = 1;

map\_0[2][1] = 1;

map\_0[0][2] = 1;

map\_0[1][2] = 1;

map\_0[2][2] = 1;

end

// FIXME: Should use system clock here.

always @(posedge clock) begin

if (pointer\_ready) begin//若监测到按键

if (pointer\_delta\_x[8]) begin//左右移动

pointer\_x <= pointer\_x - pointer\_delta\_x[7:0];

end else begin

pointer\_x <= pointer\_x + pointer\_delta\_x[7:0];

end

if (pointer\_delta\_y[8]) begin//上下移动

pointer\_y <= pointer\_y - pointer\_delta\_y[7:0];

end else begin

pointer\_y <= pointer\_y + pointer\_delta\_y[7:0];

end

end

end

assign pointer\_x\_index = pointer\_x / BLOCK\_SIZE; //目标块索引

assign pointer\_y\_index = pointer\_y / BLOCK\_SIZE;

always @(posedge clock) begin

if (enable) begin //刷新屏幕，开始运动

map\_index <= ~map\_index;

end

end

genvar i, j;

generate

for (i = 0; i < BLOCK\_COUNT\_X; i = i + 1) begin

for (j = 0; j < BLOCK\_COUNT\_Y; j = j + 1) begin

assign pointer\_select\_on[i][j] = i == pointer\_x\_index && j == pointer\_y\_index && pointer\_select; //选中目标块

//计算细胞周围生存的细胞数的数量

assign neighbor\_count\_0[i][j] = map\_0[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y] + map\_0[i][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y] + map\_0[(i + 1) % BLOCK\_COUNT\_X][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y]

+ map\_0[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][j] + map\_0[(i + 1) % BLOCK\_COUNT\_X][j]

+ map\_0[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][(j + 1) % BLOCK\_COUNT\_Y] + map\_0[i][(j + 1) % BLOCK\_COUNT\_Y] + map\_0[(i + 1) % BLOCK\_COUNT\_X][(j + 1) % BLOCK\_COUNT\_Y];

assign neighbor\_count\_1[i][j] = map\_1[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y] + map\_1[i][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y] + map\_1[(i + 1) % BLOCK\_COUNT\_X][(j - 1 + BLOCK\_COUNT\_Y) % BLOCK\_COUNT\_Y]

+ map\_1[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][j] + map\_1[(i + 1) % BLOCK\_COUNT\_X][j]

+ map\_1[(i - 1 + BLOCK\_COUNT\_X) % BLOCK\_COUNT\_X][(j + 1) % BLOCK\_COUNT\_Y] + map\_1[i][(j + 1) % BLOCK\_COUNT\_Y] + map\_1[(i + 1) % BLOCK\_COUNT\_X][(j + 1) % BLOCK\_COUNT\_Y];

always @(posedge clock) begin

if (map\_index == 0) begin //两个地图切换以实现刷新

map\_0[i][j] <= pointer\_select\_on[i][j] ^ map\_0[i][j];//生成或消除目标快内细胞

if (map\_0[i][j]) begin

map\_1[i][j] <= neighbor\_count\_0[i][j] == 2 | neighbor\_count\_0[i][j] == 3; //若原细胞为存活，周围生存细胞数为2或3，则细胞仍为生存状态

end else begin

map\_1[i][j] <= neighbor\_count\_0[i][j] == 3;//若原细胞为死亡，周围细胞数为3，则细胞变为生存状态

end

end else begin

map\_1[i][j] <= pointer\_select\_on[i][j] ^ map\_1[i][j]; //同上

if (map\_1[i][j]) begin

map\_0[i][j] <= neighbor\_count\_1[i][j] == 2 | neighbor\_count\_1[i][j] == 3;

end else begin

map\_0[i][j] <= neighbor\_count\_1[i][j] == 3;

end

end

// map\_0[i][j] <= 1;

// map\_1[i][j] <= 1;

end

end

end

endgenerate

assign x\_index = x\_position / BLOCK\_SIZE; //x轴格子索引

assign y\_index = y\_position / BLOCK\_SIZE;

assign block\_current = map\_index == 0 ? map\_0[x\_index][y\_index] : map\_1[x\_index][y\_index]; //当前格子状态

//颜色生成模块

color\_generator color\_generator(x\_index, y\_index, color\_life);

assign color\_empty = x\_index == pointer\_x\_index && y\_index == pointer\_y\_index ? COLOR\_POINTER : COLOR\_EMPTY; //不填充颜色

assign color\_block = block\_current ? color\_life : color\_empty; //填充颜色

//超出屏幕处理

color\_circlizer #(BLOCK\_SIZE) color\_circlizer (x\_position - x\_index \* BLOCK\_SIZE, y\_position - y\_index \* BLOCK\_SIZE, color\_block, color\_empty, color\_block\_circlized);

always @(\*) begin//颜色处理

if (inside\_video) begin

if (x\_position <= BLOCK\_SIZE \* BLOCK\_COUNT\_X) begin

color = circlize ? color\_block : color\_block\_circlized;

end else begin

color = COLOR\_BLACK;

end

end else begin

color = 0;

end

end

endmodule

VGA输出模块代码：

`timescale 1ns / 1ps

module vga\_controller(

input clock\_25mhz,

input reset,

output reg h\_sync,

output reg v\_sync,

output reg inside\_video,

output [9:0] x\_position,

output [8:0] y\_position

);

// SYNC, BPORCH, VIDEO, FPORCH.

parameter H\_SYNC = 96;

parameter H\_BPORCH = 144;

parameter H\_FPORCH = 784;

parameter H\_TOTAL = 800;

parameter V\_SYNC = 2;

parameter V\_BPORCH = 35;

parameter V\_FPORCH = 511;

parameter V\_TOTAL = 525;

reg [9:0] h\_counter = 0;

reg [9:0] v\_counter = 0;

reg v\_enable = 0;

always @(posedge clock\_25mhz or posedge reset) begin

if (reset) begin

h\_counter <= 0;

end else if (h\_counter == H\_TOTAL - 1) begin

h\_counter <= 0;

v\_enable <= 1;

end else begin

h\_counter <= h\_counter + 1'b1;

v\_enable <= 0;

end

end

always @(\*) begin

if (h\_counter < H\_SYNC) begin

h\_sync = 0;

end else begin

h\_sync = 1;

end

end

always @(posedge clock\_25mhz or posedge reset) begin

if (reset) begin

v\_counter <= 0;

end else if (v\_enable) begin

if (v\_counter == V\_TOTAL - 1) begin

v\_counter <= 0;

end else begin

v\_counter <= v\_counter + 1'b1;

end

end

end

always @(\*) begin

if (v\_counter < V\_SYNC) begin

v\_sync = 0;

end else begin

v\_sync = 1;

end

end

always @(\*) begin

if ((h\_counter >= H\_BPORCH) && (h\_counter < H\_FPORCH) && (v\_counter >= V\_BPORCH) && (v\_counter < V\_FPORCH)) begin

inside\_video = 1;

end else begin

inside\_video = 0;

end

end

assign x\_position = h\_counter - H\_BPORCH;

assign y\_position = v\_counter - V\_BPORCH;

endmodule

module RanGen(

input gen,

input clk,

output num

);

reg [7:0]rand\_num;

reg [7:0]ran\_num = 8'b10001101;

assign num = rand\_num[7];

always@(posedge clk or negedge gen)

begin

if(gen)

num <=0;

else

begin

rand\_num[0] <= rand\_num[7];

rand\_num[1] <= rand\_num[0];

rand\_num[2] <= rand\_num[1];

rand\_num[3] <= rand\_num[2];

rand\_num[4] <= rand\_num[3]^rand\_num[7];

rand\_num[5] <= rand\_num[4]^rand\_num[7];

rand\_num[6] <= rand\_num[5]^rand\_num[7];

rand\_num[7] <= rand\_num[6];

end

end

endmodule

`timescale 1ns / 1ps

module color\_generator(

input [4:0] x\_index,

input [4:0] y\_index,

output [11:0] color

);

assign color[11:8] = x\_index /12+9;

assign color[7:4] = y\_index / 9 + 9;

assign color[3:0] = x\_index/16+y\_index/16+7;

endmodule

`timescale 1ns / 1ps

module color\_circlizer(

input [4:0] x\_position,

input [4:0] y\_position,

input [11:0] color\_in,

input [11:0] color\_blank,

output [11:0] color\_out

);

parameter BLOCK\_SIZE = 10;

// 32 / 2 = 16

wire [3:0] x = x\_position > BLOCK\_SIZE / 2 ? x\_position - BLOCK\_SIZE / 2 : BLOCK\_SIZE / 2 - x\_position;

wire [3:0] y = y\_position > BLOCK\_SIZE / 2 ? y\_position - BLOCK\_SIZE / 2 : BLOCK\_SIZE / 2 - y\_position;

// Use < to add some space between circles.

assign color\_out = x \* x + y \* y < (BLOCK\_SIZE / 2) \* (BLOCK\_SIZE / 2) ? color\_in : color\_blank;

endmodule

**四：实验结果与分析**

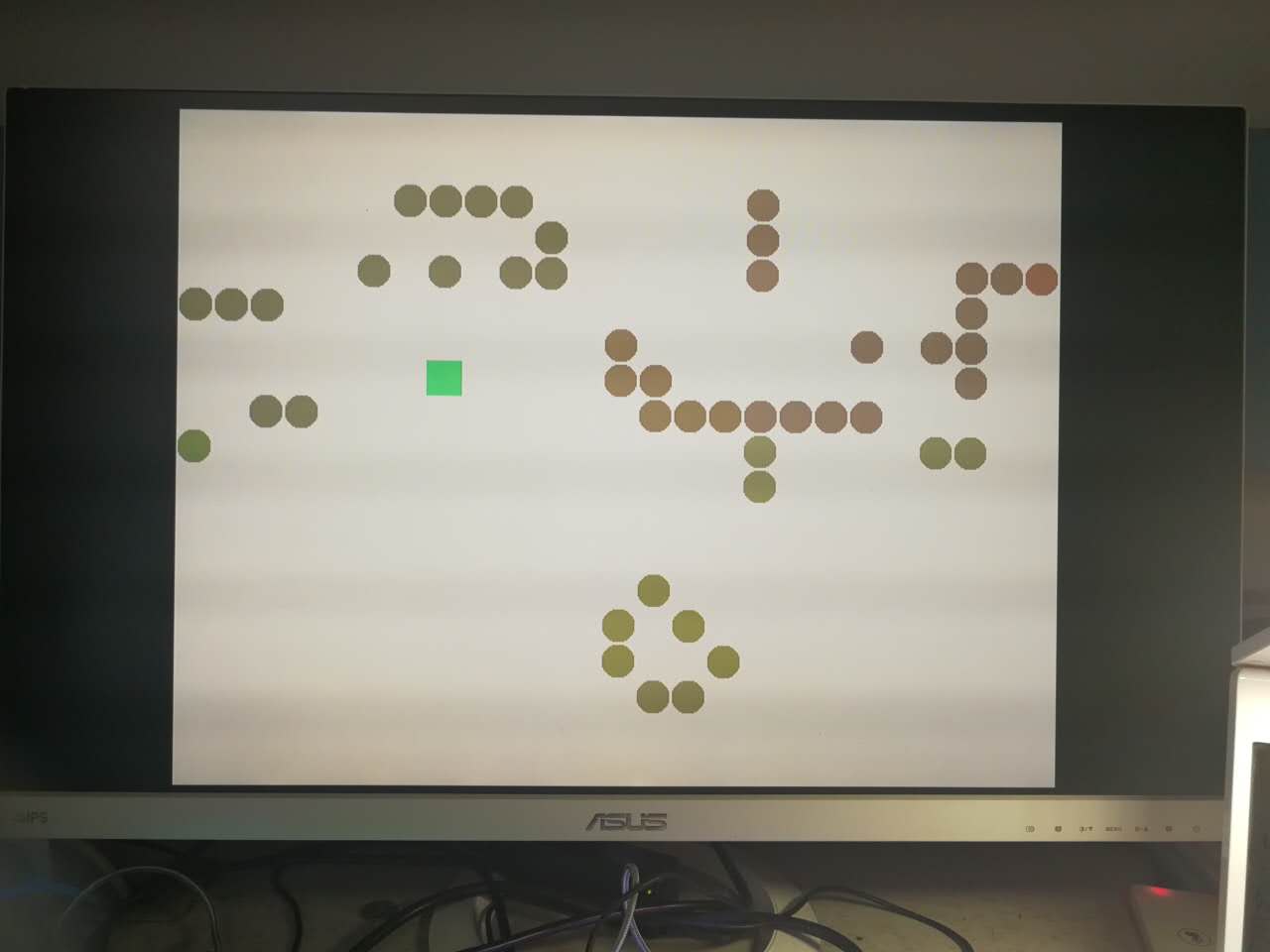


图1

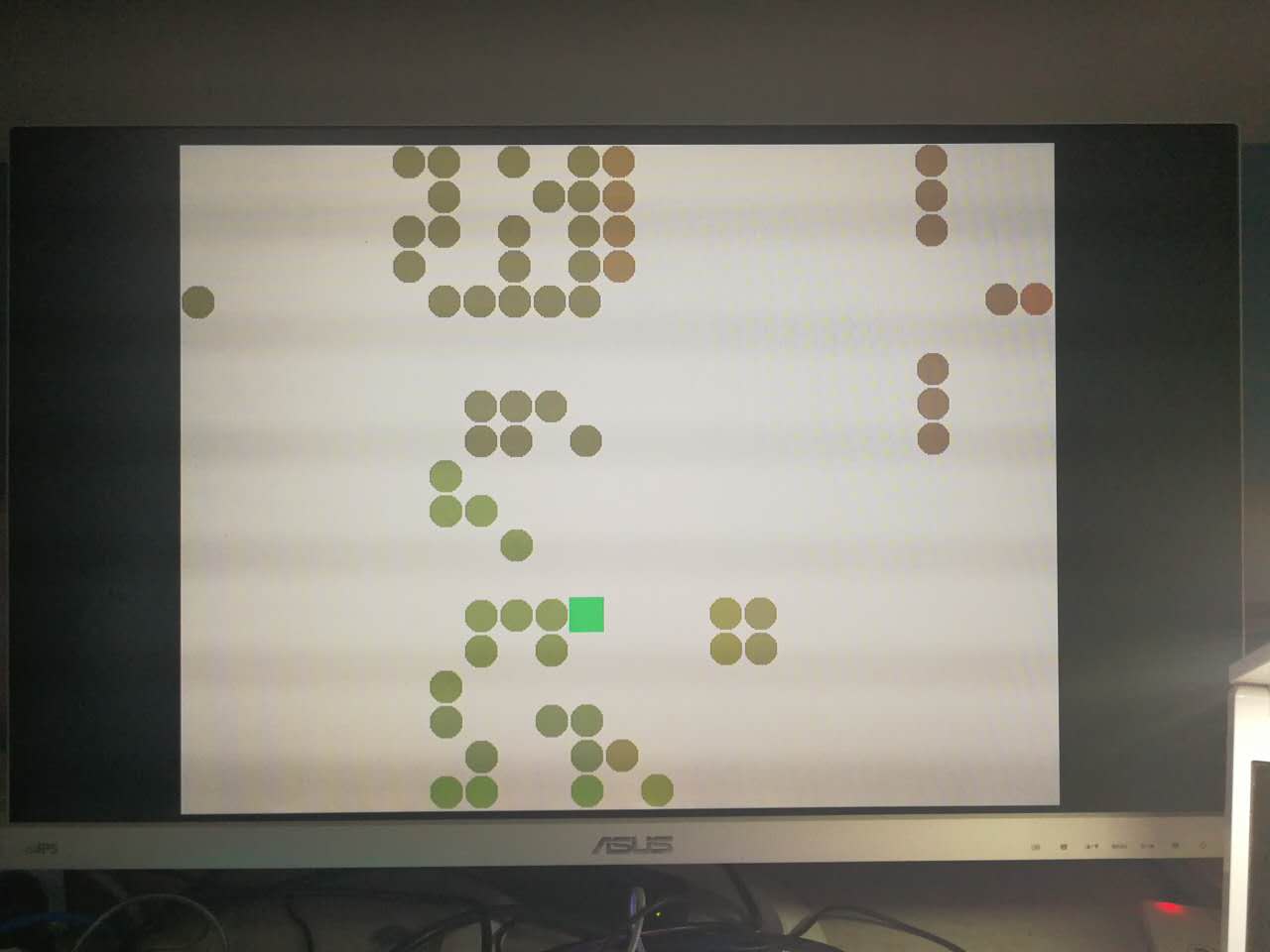


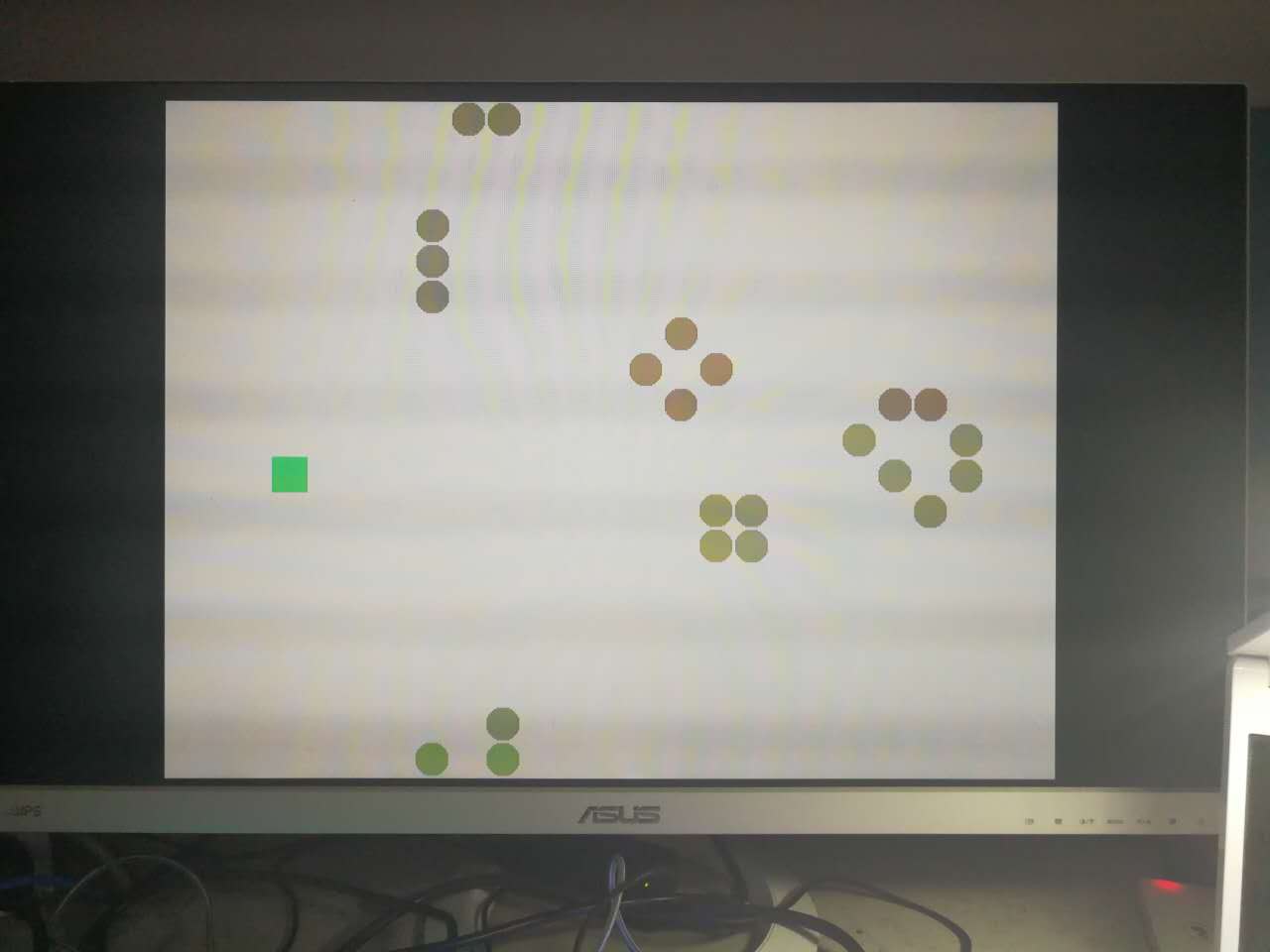
图2

图3

从图中我们可以看到，有一个绿色的方块，那就是我们的目标控制方块，我们可以对它进行移动并添加/删除其对应方格的细胞，在演化过程中，已有的细胞族群会对其它细胞族群产生影响，图1和图2表示演化中出现的情况，图3表示一些已经成型的结构，这些有特定形状的结构由他们的特点，如静物族会保持不动，震荡族在几个状态内循环，宇宙飞船族，在整个网络中飘荡，直到遇到其他细胞干扰。此外我们能够看到整个地图中，方块格的颜色会根据其位置产生微妙的不同。

**五：结论（讨论）**

**1、实验结论**

综合运用了数字逻辑课程课堂内容和前四次实验课所学知识和积累的经验，根据康威生命游戏的游戏规则，设计、并实现出了基于硬件编程的复杂数字小游戏，并使用VGA显示和用键盘进行输入。

**2、讨论**

我们最初还设想了一个功能，就是在最初随机生成细胞落，然后开始游戏，但由于时间原因，并没有实现这一功能。

其他改进还有颜色和背景的改进，以及最初游戏的开始界面。

**六、教师评审**

|  |  |
| --- | --- |
| **教师评语** | **实验成绩** |
| 签名：  日期： |  |