浙江大学

课程设计报告

中文题目 ： 迷宫小游戏

英文题目： Maze Game

姓名/学号： 卢涛/3140102441

指导教师： 施青松

参加成员： 无

专业类别： 计算机科学与技术

所在学院： 计算机学院

论文提交日期 2016 年 6 月 30 日

摘要

本文介绍的是在多周期 CPU 上运行的迷宫小游戏，即在搭建好的多周期 CPU 上，增加 VGA 控制模块，重写 MIO_BUS 模块，修改顶层模块，再写好汇编程序，写入 RAM，再执行程序，在 VGA 上显示迷宫地图，再通过用户 BTN 控制物快的上下左右移动，实现人机交互。当物块移动到迷宫出口时，则进入下一关卡，地图会随之改变。本应用设计的难点是 VGA 显示不同的地图、显示物块的移动，以及关卡的判定和切换。

关键词：MCPU、VGA 显示、地图、移动、关卡切换

目录

一、绪论 ---------- 4
1.1 设计背景 ---------- 4
1.2 国内外现状分析---------- 4
1.3 主要内容和难点---------- 4

二、设计原理---------- 5
2.1 设计相关内容---------- 5
2.2 设计方案 ---------- 5
2.3 硬件设计 ---------- 6
2.4 系统软件设计 ---------- 9

三、设计实现 ---------- 11
3.1 设计方法 ---------- 11
3.2 设计过程 ---------- 12
3.3 调试 ---------- 12

四、系统测试验证与结果分析---------- 13
4.1 功能测试 ---------- 13
4.2 结果分析 ---------- 14
4.3 系统演示与操作说明 ---------- 14

五、结论与展望---------- 14

第1章 绪论

1.1 迷宫游戏设计背景

迷宫游戏是一项非常经典的小游戏,可以考察玩家对错综复杂的迷宫地图的路径识别能力。该迷宫游戏的实现平台是多周期CPU，基本能实现迷宫游戏的要求，即显示迷宫地图，对物快进行移动的操作，对是否到达迷宫出口的判断，以及关卡之间的切换。特点是简单易玩，可玩性、思考性较强。

1.2 国内外现况分析

本设计是基于本学期我们自己写的CPU，而这一块，在国内，浙大是领先的，因此，基于MCPU上的迷宫游戏在国内外还是不多的。但是，迷宫游戏在其他平台上的实现还是非常多的。本设计与它们的区别就在于平台的不同，但是在实现功能上的差异不大，可玩性也区别不大。

1.3 主要内容和难点

本设计的主要内容：

1、 实现VGA的控制模块，对应的MIO_BUS以及顶层模块的修改。
2、 实现VGA上迷宫地图的显示。
3、 通过BTN对物快进行移动和判定。
4、 对关卡结束的判定以及关卡的切换。

技术要求：要求对多周期CPU构架的深刻理解，对VGA显示原理的熟练掌握，以及对MIP汇编语言的熟练运用。

实现的重点和难点：
VGA的显示、汇编语言对游戏的控制。

第2章 迷宫游戏设计原理

2.1 迷宫游戏设计相关内容

主要理论：VGA 显示原理

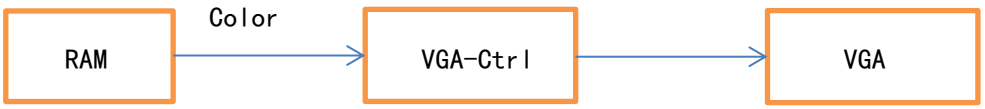
VGA 行列为 640*480，由于存储空间的不足，所以设置为 80*60 的像素，并且根据行列信号逐行、逐列显示。

主要平台：

多周期 CPU,支持 MIPS15 条指令。

2.2 迷宫游戏设计方案

首先，解决硬件上的问题。增加 VGA 控制模块，将 RAM 改成双端口，直接把 VRAM 放在 RAM 里面，让 VGA 直接从 RAM 读取 RGB 值，从而让 VGA 显示地图。



图一：VGA 显示原理

在软件上，写好汇编程序，翻译成 COE，再导入 CPU 中的 RAM，通过程序改变 RAM 中的值，从而改变 VGA 显示的内容，实现移动、关卡切换等功能。

图二：实现具体功能框图

2.3 迷宫游戏硬件设计

顶层模块代码：

RAM 采用双端口

```
    RAM_B   U3(
    .addra(addra), .wea(wea), .dina(dina), .clka(clk_100mhz), .douta(dout
a), .addrb(addrb), .web(1'b0), .clkb(clk_100mhz), .doutb(doutb)
    );
```

VGA_ctrl:

```
    VGA_ctrl    U11(
```

```
    .clk(clk_VGA), .rst(rst), .hs(HSYNC), .vs(VSYNC), .addrb(addrb),
            .color(doutb), .red(Red), .green(Green), .blue(Blue)
    );
```

VGA_ctrl 代码:

```
module VGA_ctrl(
    input clk,
    input rst,
    output hs,
    output vs,
    output [3:0] red,
    output [3:0] green,
    output [3:0] blue,
    input [31:0] color,
    output [9:0] x_ptr,
    output [9:0] y_ptr,
    output reg[13:0] addrb,
    output valid
);
    reg [9:0] cnt_x;
    reg [9:0] cnt_y;
    reg  judge;

    initial begin
    cnt_x <= 10'b0;
    cnt_y <= 10'b0;
    judge <= 0;
    end

    always @*begin
         if(color == 32'hffff0000)
          judge <= 1;
    end

    always  @* begin
         if(judge == 0)
          addrb <= valid ? (14'd2048 + {1'b0,y_ptr[9:3], 6'b0} + {3'b0, y_ptr[9:3],
4'b0} + {7'b0, x_ptr[9:3]}) : 14'b0;
          else
          addrb <= valid ? (14'd6848 + {1'b0,y_ptr[9:3], 6'b0} + {3'b0, y_ptr[9:3],
4'b0} + {7'b0, x_ptr[9:3]}) : 14'b0;
    end
```

```
    always @(posedge clk) begin
        if (rst) cnt_x <= 0;
        else if (cnt_x == 10'd799) cnt_x <= 0;
        else cnt_x <= cnt_x + 1;
    end

    always @(posedge clk) begin
        if (rst) cnt_y <= 0;
        else if(cnt_x == 10'd799) begin
            if (cnt_y == 10'd524) cnt_y <= 0;
            else cnt_y <= cnt_y+1;
        end
    end

    assign hs = !((cnt_x >= 0) && (cnt_x < 96));
    assign vs = !((cnt_y >= 0) && (cnt_y < 2));
    assign valid = (cnt_x >= 96+40+8) && (cnt_x < 96+40+8+640) && (cnt_y >= 2+25+8)
&& (cnt_y < 2+25+8+480);

    assign red = valid ? color[11:8] : 0;
    assign green = valid ? color[7:4] : 0;
    assign blue = valid ? color[3:0] : 0;
    assign x_ptr = cnt_x - (96+40+8);
    assign y_ptr = cnt_y - (2+25+8);
endmodule
```

MIO_BUS 代码:

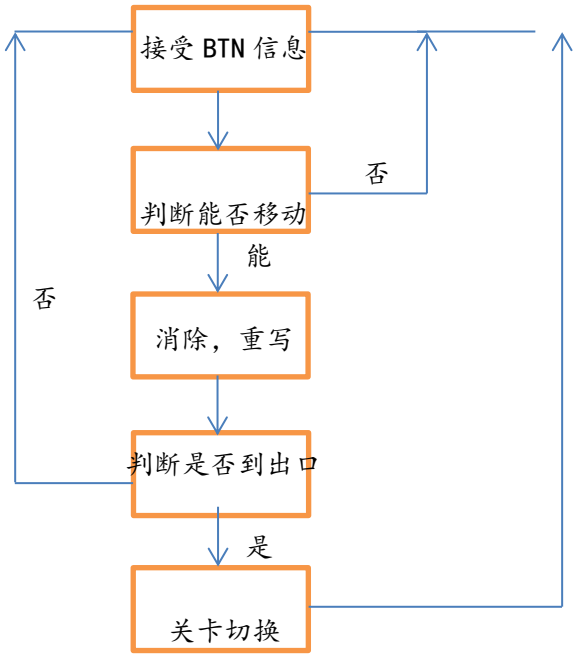
```
module MIO_BUS(
    input clk,
    input rst,
    input[3:0]BTN,
    input[15:0]SW,
    input mem_w,
    input[31:0]Cpu_data2bus,                //data from CPU
    input[31:0]addr_bus,
    input[31:0]ram_data_out,
    input[15:0]led_out,
    input[31:0]counter_out,
    input counter0_out,
    input counter1_out,
    input counter2_out,

    output reg[31:0]Cpu_data4bus,              //write to CPU
```

```
    output reg[31:0]ram_data_in,                //from CPU write to Memory
    output reg[13:0]ram_addr,                       //Memory Address signals
    output reg data_ram_we,
    output reg GPIOf0000000_we,
    output reg GPIOe0000000_we,
    output reg counter_we,
    output reg[31:0]Peripheral_in
);
    reg data_ram_rd,GPIOf0000000_rd,GPIOe0000000_rd,counter_rd;

    always @ * begin
        data_ram_we = 0;
        data_ram_rd = 0;
        counter_we = 0;
        GPIOf0000000_we = 0;
        GPIOf0000000_rd = 0;
        GPIOe0000000_we = 0;
        GPIOe0000000_rd = 0;
        counter_rd = 0;
        ram_addr = 13'h0;
        ram_data_in = 32'h0;
        Peripheral_in = 32'h0;
        case(addr_bus[31:28])
            4'h0:begin
                 data_ram_we = mem_w;
                 ram_addr = addr_bus[15:2];
                 ram_data_in = Cpu_data2bus;
                 data_ram_rd = ~mem_w;
            end

            4'he:begin
                GPIOe0000000_we = mem_w;
                Peripheral_in = Cpu_data2bus;
                GPIOe0000000_rd = ~mem_w;
            end

            4'hf:begin
                if(addr_bus[2])begin
                    counter_we = mem_w;
                    Peripheral_in = Cpu_data2bus;
                    counter_rd = ~mem_w;
                end
                else begin
                    GPIOf0000000_we = mem_w;
```

```
                    Peripheral_in = Cpu_data2bus;
                    GPIOf0000000_rd = ~mem_w;
                end
            end
        endcase
    end

    always @ * begin
        casex({data_ram_rd,GPIOe0000000_rd,counter_rd,GPIOf0000000_rd})
            4'b1xxx:Cpu_data4bus = ram_data_out;
            4'bx1xx:Cpu_data4bus = counter_out;
            4'bxx1x:Cpu_data4bus = counter_out;
            4'bxxx1:Cpu_data4bus =
{counter0_out,counter1_out,counter2_out,17'b0,BTN[3:0],SW[7:0]};
            default :Cpu_data4bus = 32'h0;
        endcase
    end
endmodule
```

2.4 系统软件设计

用两个寄存器保存当前木块的横纵坐标，根据读入的 BTN 以及周围墙的情况判断是否能够移动，若移动，则改变 RAM 中相应的 COLOR 值，若物块坐标与出口坐标相同，则自动切换到下一关卡，即改变 VGA 中接受的初始地址值。



接受BTN实现移动代码：

```
loop:
jal  delay;                          //delay
lw   $s5,0($s7);                     //读取BTN信息
sw   $s5,0($a2);
addi $s4,$zero,0x100;                //s4 = 0x100;
and  $t1,$s5,$s4;
bne  $t1,$zero,goleft;               //左移
lw   $s5,0($s7);
addi $s4,$zero,0x200;
and  $t1,$s5,$s4;
bne  $t1,$zero,goright;              //右移
lw   $s5,0($s7);
addi $s4,$zero,0x400;
and  $t1,$s5,$s4;
bne  $t1,$zero,goup;                 //上移
lw   $s5,0($s7);
addi $s4,$zero,0x800;
and  $t1,$s5,$s4;
bne  $t1,$zero,godown;               //下移
```

左移代码：

```
goleft:
addi $s0,$s0,-4;
add  $t0,$s0,$s1;
addi $t0,$t0,8520;
lw   $t1,0($t0);
beq  $t1,$s6,back;                   //判断能否左移，若不能，能返回
lw   $t1,4($t0);
beq  $t1,$s6,back;
lw   $t1,320($t0);
beq  $t1,$s6,back;
lw   $t1,324($t0);
beq  $t1,$s6,back;
addi $s0,$s0,4;                      //如果可以左移，则先把当前位置的颜色改为背景色
add  $t0,$s0,$s1;
addi $t0,$t0,8520;
sw   $zero,0($t0);
sw   $zero,4($t0);
sw   $zero,320($t0);
sw   $zero,324($t0);
addi $s0,$s0,-4;
add  $t0,$s0,$s1;
```

```
addi $t0,$t0,8520;
sw   $a3,0($t0);                    //再把下一位置的颜色改为红色，实现左移效果
sw   $a3,4($t0);
sw   $a3,320($t0);
sw   $a3,324($t0);
j    judge;
back:
addi $s0,$s0,4;
j judge;                            //判断
```

第3章 迷宫游戏设计实现

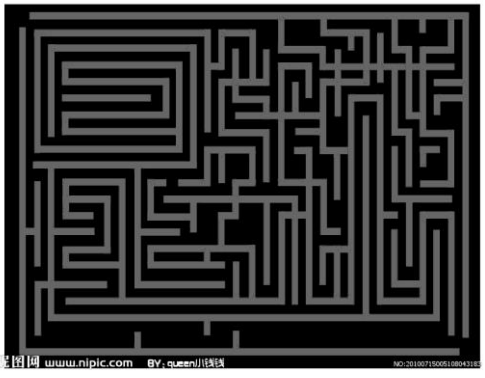
3.1 实现方法

1、 顶层模块：加入 VGA_ctrl 模块，同时把 RAM 改为双端口。

2、 VGA_ctrl 模块：直接把 VRAM 放在 RAM 中，VGA_ctrl 从 RAM 中直接取值。第一个点从 RAM 中的第 2048 个位置取值,接下去的 8*8 个方格内的像素点颜色都为这个值，从而减少存储量。然后，随着 VGA 的不断扫描显示，所对应的地址值也不断+1，显示不同的颜色，从而实现显示部分的功能。

3、 物块的移动、判定、关卡的切换全部由汇编程序来实现。物体的移动原理是，接受 BTN 信息，获取相对应的操作，若没有墙的阻挡，则移动，移动就是把当前位置在 RAM 所对应的 COLOR 值改为背景色,再将下一位置的颜色改为红色,从而实现移动。再判断是否已到达迷宫出口,如果已到达,则在 VGA 控制中改变接收地址的初始值，显示下一张地图，重复循环，直至游戏结束。

重点难点：

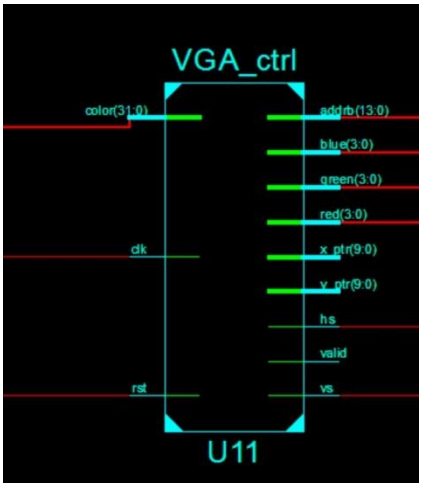


图三：迷宫图

将图片用 MATLAB 进行处理，得到每个点的 RGB 值，再将其一次存储到 RAM 中，再让 VGA 从 RAM 读取，从而实现显示，每当地图切换时，VGA 接收地址的初值也将随之改变。

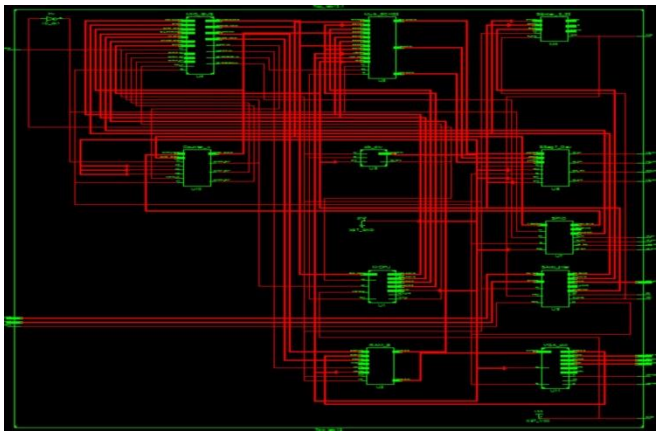
3.2 实现过程

先实现 VGA 控制模块，以下为其 RTL 图：



图四：VGA_CTRL RTL 图

再实现顶层模块，以下为其 RTL 图：



图五：TOP RTL 图

最后，写好汇编程序，导入 RAM，实现所有功能。

3.3 调试

调试中出现的问题及解决：

一开始,VGA 无法显示图像,后来发现是 VGA 控制模块的显示边界条件出现了错误。后来当 VGA 可以正常显示时，发现并没有显示出存储的地图信息，后来发现是 VGA 显示存在偏差，经过对接收地址的改变后，错误也就解决了。

后来，又遇到物块移动过快的问题，发现是没有 delay，导致按一下 BTN，物块会移动很多格，于是，加了 delay 函数之后，问题就解决了。

最后，遇到的问题是无法进行关卡的切换，后来发现是 VGA 控制模块的判定出了问题，修改之后，整个系统就不存在问题了。

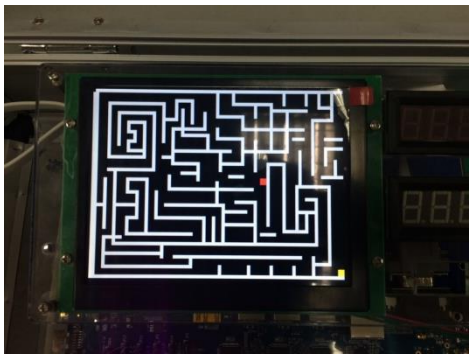
第4章 系统测试验证与结果分析

4.1 功能测试

初始地图显示：

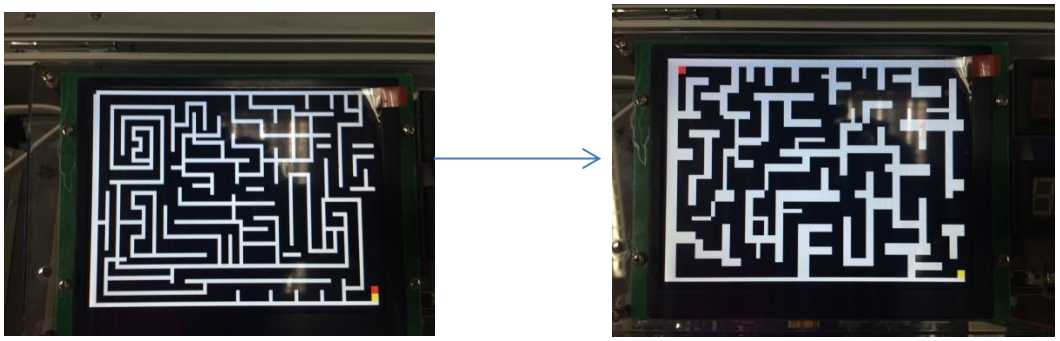
图六：初始地图显示

按下 RST 键，游戏开始。移动过程中的地图显示：



图七：游戏过程中的地图显示

到达出口后的地图切换；



图八：将要到达出口显示图　　图九：切换后的地图显示

4.2 结果分析

经测试，系统的地图显示，物块的判断是否能移动、移动功能、判断是否到达出口、关卡的切换功能全部正确。

4.3 系统演示与操作说明

按下 RST 键，游戏开始，按下 BTN 控制物快移动，从左往右依次为左、右、上、下。当全部地图都走完时，物块将无法移动，此时游戏结束。

第5章 结论与展望

在做这次 project 的过程中，由于我上学期选的不是施老师的课，因此对 VGA 这一块是完全没有接触的，所以我在 VGA 的显示上，花费了大量的时间。从一开始的什么都不显示，到后来的显示内容与 RAM 中的值不对应，再到后来关卡不能切换，我遇到了很多的问题。不过在完全搞清楚 VGA 的显示原理之后，在和同学的讨论下以及自己的思考下， 我最终还是完成了这次的大程。

做完这次大程，我觉得我对 CPU 的构造有了更深刻的理解，对 VGA 的显示原理有了进一步的认识，对于汇编语言的编写也更加地熟练，另外，更重要的是，我觉得我的动手能力、独立思考能力在这一过程中都得到了大大的提升，这对我的帮助已经远远超过了一个大程，受益匪浅。

对教学的建议的话，我觉得施老师的实验课上的非常的好，但是理论课的话，我觉得施老师可以多和我们进行互动，这样课堂效率会高一点，总的来说，上了施老师的课，收获还是很大的，感谢施老师，感谢助教！