课程设计报告

中文题目 ： 基于数字系统的方块跑酷游戏

英文题目： Block Running Based on the Digital System

姓名/学号： 张倬豪 3150102418

指导教师： 施青松

参加成员： 无

专业类别： 求是科学班（计算机）

所在学院： 计算机学院

论文提交日期 2017 年 6 月 18 日

摘要

《方块跑酷》是一款仿照 Flappy Bird、是男人就下一百层等简单的横板过关游戏而开发的躲避类跑酷游戏，其画面采取像素风格，方块必须躲避不断下落的横杠，通过缝隙，计分加 1，否则撞到横杠即为失败。本次课程设计基于数字系统开发，在计算机组成实验 12（多周期 CPU）的基础上进行扩展，因为是兴趣小组成员，还加入了 VGA、PS2 的扩展设计，采用 VRAM（显存）实现 VGA，通过 MIO_BUS 实现 PS2 信号接入，在 RAM_B 中实现 MIPS 汇编指令控制游戏逻辑和写入显存。

关键词：跑酷躲避游戏，FGPA，VGA，PS2，Multi-CPU

目录

第1章 绪论

1.1 方块跑酷游戏设计背景

通过计算机组成课程的学习，对硬件设备的了解已经超越了上学期的数逻实验，本次课程设计的思想基于整个计算机组成课程的设计思路，基于前面完成的实验单周期或者多周期的 CPU，扩展总线控制器的 VGA、PS2 接口，并且基于这些设计完成一个小型应用程序。意义在于加深了对 CPU 以及总线和外部输入输出设备间交互原理的理解，并通过这次设计练习使用 MIPS 汇编指令编写程序，为后续课程做知识储备。

游戏特点在于工程中同时整合和体现了**基本七段数码管、SW 开关、基于显存的 VGA、PS2** 等所有上课提到过的模块知识，有很好的用户交互性，而且游戏也具有一定的可玩性。虽然还有一些不足之处可以改进，但是总体达到了课程设计要求的目的。

1.2 主要内容和难点

主要内容包括：

1. 监听玩家对按键的交互反应，通过按键控制方块的左右移动来躲避不断下降的横杠。
2. 控制横杠在玩家不断操纵的同时仍然以一定的速度下降。
3. 能够进行基本的碰撞检测，判断横杠与方块的碰撞和穿越，如果碰撞则跳至重新开始，分数也因此清零，如果成功穿过横杠则计分+1。

主要难点包括：

1. 按键的输入如何进行基本的处理，并通过接口进入总线控制，分配地址，然后被 CPU 读取等。
2. CPU、VGA 的时序问题，需要详细考虑各个模块的时序，防止游戏的实现过程发生停顿或者刷新过快的情况，当时序确定的情况下如何进行汇编指令的调整来使游戏变得可玩。
3. VGA 的 VRAM 显存实现，计算机组成课程不同与数字逻辑的地方就在于 CPU 的实现，因此我们绝不能仅仅把 CPU 当成一个 ALU 计算器进行一些基本的坐标运算，而将 VGA 与地址、数据的交互发生在外部，比如用一个辅助模块进行 VGA 的交互就十分不可取，应当用一个显存来实现 VGA 的交互，汇编指令根据玩家输入通过 CPU 仅进行显存的写操作，VGA 模块对显存仅进行读取。
4. 随机数的实现，本次工程采用取某一时钟加上用户输入的干扰来达到随机数的效果，在用户不断操作的时候效果良好，可是在用户停止时随机数会不断产生为同一个，暂时没有想到更好的算法。

第2章 方块跑酷游戏设计原理

2.1 游戏设计相关内容

本课程设计中，使用了多周期 CPU 设计实验中最后一次实验—指令扩展实验后的 CPU，在这个基础上添加简易的总线和外设。

整个游戏的设计原理设计总线的信号传递，VGA 时序及原理

VGA 原理

显示器扫描方式为逐行扫描：逐行扫描是扫描从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。

完成一次行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率。

显示带宽

带宽指的显示器可以处理的频率范围。如果是 60Hz 刷新频率的 VGA,其带宽达 640x480x60=18.4MHz。

时钟频率

使用 640x480@59.94Hz(60Hz)，每场对应 525 个行周期

(525=10+2+480+33),其中 480 为显示行。每场有场同步信号,该脉冲宽度为 2 个行周期的负脉冲，每显示行包括 800 点时钟,其中 640 点为有效显示区,每一行有一个行同步信号，该脉冲宽度为 96 个点时钟。由此可知：行频为 525*59.94=31469Hz,需要点时钟频率：525*800*59.94 约 25MHz。

PS2 输入原理

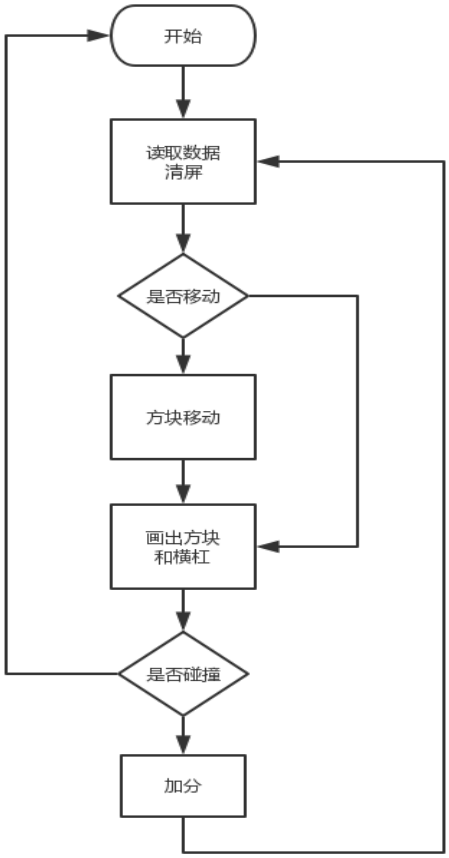
PS2 按键在未被按下的时候，处于低电平，按下之后处于高电平，由于按键通过物 理传动，可能因为接触问题产生抖动，使得按键的输入方波变为一个主方波和多个 附加的方波，可能对后续判断产生影响，所以在按键输入时应该先进行一次按键去抖动。

MIO_BUS 工作原理

MIO_BUS 是 CPU 与外部数据交换接口模块，本课程实验将数据交换电路合并成一 个地址译码电路

提供 CPU 读取 SW、BTN 等外设的数据以及读取数据存储内容,向外设 7-Seg、LED 等接口传送信号的功能，控制 CPU 与外部数据交换。

2.2 游戏设计方案



如图，游戏流程图就是简单的一个无限循环。

2.3 硬件设计

（内容要点：详细的硬件设计分析过程，包括顶层及各电路模块结构、逻辑电路图、硬件描述代码等）

硬件设计在之前的实验中已经多次提及，本次完全采用了以前的实验框架Framework.v，以及多周期CPU。其中需要修改顶层代码、MIO_BUS代码、加入VGA代码、PS2代码、CLK模块代码他们的设计如下：

CLK模块，加入了随机数的生成：

```
module clk_div(
       input clk,
       input rst,
       input SW2,
       output reg [31:0]clkdiv,
       output Clk_CPU,
       output reg [10:0]num
    );
```

```
    initial
     begin
        num <= 0;
     end

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            clkdiv <= 0;
        end else begin
            clkdiv <= clkdiv + 1'b1;
            num <= (num + 1) % 59;
        end
    end

    assign Clk_CPU = (SW2)?clkdiv[24]:clkdiv[2];
endmodule
```

VGA 模块，采用 640x480 分辨率：

```
module vga_controller(
    input clk, //25mhz
    input rst,
    input [11:0] d_in,
    output reg rdn,
    output reg [3:0] r,
    output reg [3:0] g,
    output reg [3:0] b,
    output reg hs,
    output reg vs,
    output [12:0] vram_addr //vram的地址
    ); // vgac
    wire [9:0] row, col;
    // h_count: VGA horizontal counter (0-799)
    reg [9:0] h_count; // VGA horizontal counter (0-799): pixels
    always @ (posedge clk) begin
        if (rst) begin
            h_count <= 10'h0;
        end else if (h_count == 10'd799) begin
            h_count <= 10'h0;
        end else begin
            h_count <= h_count + 10'h1;
        end
    end
    // v_count: VGA vertical counter (0-524)
```

```
    reg [9:0] v_count; // VGA vertical  counter (0-524): lines
    always @ (posedge clk or posedge rst) begin
        if (rst) begin
            v_count <= 10'h0;
        end else if (h_count == 10'd799) begin
            if (v_count == 10'd524) begin
                v_count <= 10'h0;
            end else begin
                v_count <= v_count + 10'h1;
            end
        end
    end
    // signals, will be latched for outputs
    assign     row    =  v_count - 10'd35;     // pixel ram row addr
    assign     col    =  h_count - 10'd143;    // pixel ram col addr
    wire       h_sync = (h_count > 10'd95);    //  96 -> 799
    wire       v_sync = (v_count > 10'd1);     //   2 -> 524
    wire       read   = (h_count > 10'd142) && // 143 -> 782
                        (h_count < 10'd783) && //       640 pixels
                        (v_count > 10'd34)  && //  35 -> 514
                        (v_count < 10'd515);   //       480 lines
    // vga signals
    always @ (posedge clk) begin
        //row_addr <=  row[8:0]; // pixel ram row address
        //col_addr <=  col;     // pixel ram col address
        rdn     <= ~read;     // read pixel (active low)
        hs      <=  h_sync;   // horizontal synchronization
        vs      <=  v_sync;   // vertical  synchronization
        r       <=  rdn ? 4'h0 : d_in[11:8]; // 3-bit red
        g       <=  rdn ? 4'h0 : d_in[7:4]; // 3-bit green
        b       <=  rdn ? 4'h0 : d_in[3:0]; // 3-bit blue
    end
    assign vram_addr = rdn ? 13'h0 : (/*13'd1024 + */{row[9:3], 6'h0} +
{2'h0, row[9:3], 4'h0} + {6'h0, col[9:3]});
endmodule
```

PS2，仅给出两个信号（左右）接口，方便使用：

```
module keyboard(
    input clk25,
    input clr,
    input PS2C,
    input PS2D,
    output reg left,
    output reg right
```

```
);

    reg PS2Cf, PS2Df;
    reg [7:0] ps2c_filter, ps2d_filter;
    reg [10:0] shift1, shift2;

    always @(posedge clk25)
    begin
        ps2c_filter[7] <= PS2C;
        ps2c_filter[6:0] <= ps2c_filter[7:1];
        ps2d_filter[7] <= PS2D;
        ps2d_filter[6:0] <= ps2d_filter[7:1];
        if (ps2c_filter == 8'b11111111)
        begin
            PS2Cf <= 1;
        end
        else
        if (ps2c_filter == 8'b00000000)
        begin
            PS2Cf <= 0;
        end
        if (ps2d_filter == 8'b11111111)
        begin
            PS2Df <= 1;
        end
        else
        if (ps2d_filter == 8'b00000000)
        begin
            PS2Df <= 0;
        end
    end

    always @(negedge PS2Cf)
    begin
        shift1 <= {PS2Df, shift1[10:1]};
        shift2 <= {shift1[0], shift2[10:1]};
    end

always @(posedge clk25)
begin
    if (shift2[8:1] != 8'hF0 && shift1[8:1] == 8'h1C)
        left <= 1;
    else
        left <= 0;
```

```
    if (shift2[8:1] != 8'hF0 && shift1[8:1] == 8'h23)
        right <= 1;
    else
        right <= 0;
end

endmodule
```

MIO_BUS，实现了总线控制，之前已经阐述清楚，此处是具体实现：

```
`timescale 1ns / 1ps

module MIO_BUS(
        input [10:0] num,
        input [3:0] BTN,  // 4
        input [7:0] SW,   // 8
        input mem_w,  //  CPU
        input [31:0] Cpu_data2bus,  // CPU
        input [31:0] addr_bus,   //  CPU
        input [31:0] ram_data_out,  //  RAM
        input [7:0] led_out, //  LED
        input [31:0] counter_out,   //
        input counter0_out,  //  0
        input counter1_out,  //  1
        input counter2_out,  //  2
        output reg [31:0] Cpu_data4bus, // CPU , CPU
        output reg [31:0] ram_data_in,  // RAM  RAM
        output reg [9:0] ram_addr,  // RAM  RAM
        output reg data_ram_we,  // RAM  RAM
        output reg GPIOf0000000_we, //  LED
        output reg GPIOe0000000_we, //  7 U5
        output reg GPIOd0000000_we,
        output reg counter_we,   //  U10
        output reg [31:0] Peripheral_in,   // ,

        input ps2_left,
        input ps2_right,
        output reg vram_write,
        output reg [12:0] cpu_vram_addr,
        output reg [11:0] cpu_vram_data
    );

    reg data_ram_rd;
    reg GPIOf0000000_rd;
    reg GPIOe0000000_rd;
```

```
reg counter_rd;
reg [10:0] rand;

always @(*) begin

    data_ram_we=0;
    data_ram_rd=0;
    counter_we=0;
    counter_rd=0;
    GPIOf0000000_we=0;
    GPIOe0000000_we=0;
    GPIOf0000000_rd=0;
    GPIOe0000000_rd=0;
    ram_addr=10'h0;
    ram_data_in=32'h0;
    Peripheral_in=32'h0;
    Cpu_data4bus =32'h0;

    case (addr_bus[31:28])
        4'h0: begin
            if (addr_bus == 32'h0000xxxx) begin
                data_ram_we = mem_w;
                ram_addr = addr_bus[11:2];
                ram_data_in = Cpu_data2bus;
                Cpu_data4bus = ram_data_out;
                data_ram_rd = ~mem_w;
            end
            else if (addr_bus == 32'h000cxxxx) begin
                vram_write = mem_w;
                cpu_vram_addr = addr_bus[12:0];
                cpu_vram_data = Cpu_data2bus[11:0];
            end
            else if (addr_bus == 32'h000d0100) begin //PS2 left(A)
                Cpu_data4bus = {31'h00000000,ps2_left};
            end
            else if (addr_bus == 32'h000d0101) begin // PS2 right(D)
                Cpu_data4bus = {31'h00000000,ps2_right};
            end
            else if (addr_bus == 32'h000d0102) begin // random
                if (ps2_left || ps2_right) rand = (num + 32) % 59;
                Cpu_data4bus = {21'h000000,rand};
            end
        end
```

```
            4'he: begin    // 7 segments LEDs
                GPIOe0000000_we = mem_w;
                Peripheral_in = Cpu_data2bus;
                Cpu_data4bus = counter_out;
                GPIOe0000000_rd = ~mem_w;
            end
            4'hf: begin
                if (addr_bus[2]) begin
                    counter_we = mem_w;
                    Peripheral_in = Cpu_data2bus;
                    Cpu_data4bus = counter_out;
                    counter_rd = ~mem_w;
                end else begin
                    GPIOf0000000_we = mem_w;
                    Peripheral_in = Cpu_data2bus;
                    Cpu_data4bus = {counter0_out, counter1_out,
counter2_out, 9'h00, led_out, BTN, SW};
                    GPIOf0000000_rd = ~mem_w;
                end
            end
        endcase

    end

endmodule
```

Framework（相关部分），各模块之间的联系之前已经阐述清楚，此处是具体实现：

```
RAM_B  XLXI_13 (.addra(ram_addr[9:0]),
                .clka(clk_100mhz),
                .dina(ram_data_in[31:0]),
                .wea(data_ram_we[0]),
                .douta(ram_data_out[31:0]));

   MCPU  XLXI_30 (.clk(Clk_CPU),
                           .Data_in(Data_in[31:0]),
                           .INT(counter0_OUT),
                           .MIO_ready(V5),
                           .reset(rst),
                           .Addr_out(Addr_out[31:0]),
                           .CPU_MIO(),
                           .Data_out(Data_out[31:0]),
                           .inst_out(inst[31:0]),
                           .mem_w(mem_w),
```

```
                        .PC_out(PC[31:0]),
                        .state(State[4:0]));

MIO_BUS  XLXI_21 (.num(num),
                    .addr_bus(Addr_out[31:0]),
              .BTN(BTN_OK[3:0]),
              .counter_out(Counter_out[31:0]),
              .counter0_out(counter0_OUT),
              .counter1_out(counter1_OUT),
              .counter2_out(counter2_OUT),
              .Cpu_data2bus(Data_out[31:0]),
              .led_out(LED_out[15:0]),
              .mem_w(mem_w),
              .ram_data_out(ram_data_out[31:0]),
              .SW(SW_OK[15:0]),
              .counter_we(counter_we),
              .Cpu_data4bus(Data_in[31:0]),
              .data_ram_we(data_ram_we[0]),
              .GPIOe0000000_we(GPIOE0),
              .GPIOf0000000_we(GPIOF0),
                    .GPIOd0000000_we(GPIOD0),
              .Peripheral_in(GPU2IO[31:0]),
              .ram_addr(ram_addr[9:0]),
              .ram_data_in(ram_data_in[31:0]),
                    .ps2_left(signal_left),
                    .ps2_right(signal_right),
                    .vram_write(vram_we),
                    .cpu_vram_addr(cpu_vram_addr),
                    .cpu_vram_data(cpu_vram_data)
                    );

VRAM  U01 (       .addra(cpu_vram_addr),
                  .dina(cpu_vram_data),
                  .wea(vram_we),
                  .clka(Div[0]),
                  .addrb(vram_addr),
                  .clkb(Div[0]),
                  .doutb(vram_out)
                  );

assign clock_vga = Div[1];
assign clock_ps2 = Div[1];
vga_controller U00 (
    .clk(clock_vga),
```

```
        .rst(rst),
        .hs(h_sync),
        .vs(v_sync),
        .rdn(inside_video),
        .vram_addr(vram_addr),
        .d_in(vram_out),
        .r(Red),
        .g(Green),
        .b(Blue)
    );

    keyboard U02 (
        .clk25(clock_ps2),
        .clr(rst),
        .PS2C(PS2C),
        .PS2D(PS2D),
        .left(signal_left),
        .right(signal_right)
    );
```

2.4 系统软件设计（若有）

（内容要点：详细的软件设计分析过程，包括测试代码，主程序和各功能子模块代码、软件框图和流程图等）

汇编指令本身可读性非常低，因此汇编指令中均加入了大量注释来帮助读者理解：

```
Initial:
lui $s6, 0x000D; // Address: 0x000Dxxxx
lui $s7, 0x000D; // Address: 0x000Dxxxx
lui $s5, 0x000D; // Address: 0x000Dxxxx
addi $s6, $s6, 256; // PS2: left signal's address
addi $s7, $s7, 257; // PS2: right signal's address
addi $s5, $s5, 258; // random number's address
addi $s0, $zero, 40; // Block's x(initialized as 40)
addi $s1, $zero, 0; // Barrier's y(initialized as 0)
addi $s2, $zero, 79; // Max_x
addi $s3, $zero, 59; // Max_y
and $s4, $s4, $zero // Score
lui $t1, 0xE000; // Address: 0xE000(7 segment)
sw $s4, 0($t1); // Store score
lw $t5, 0($s5); // random number

Start:
lui $t1, 0x000c;
```

```
addi $t1, $t1, 0x12bf; // t1: the end of VRAM's address
lui $t2, 0x000c; // t2: the beginning of VRAM's address
addi $t3, $zero, 0xF0; // t3: Green color
Loop:
sw $t3, 0($t2); // draw at t2
addi $t2, $t2, 1; // next address of t2
slt $t4, $t1, $t2; // if t2 is less than t1, loop
beq $t4, $zero, Loop;

addi $t4, $zero, 0;
addi $t3, $zero, 500;
Delay1:
addi $t4, $t4, 1;
bne $t4, $t3, Delay1; // delay 500 times for vram to catch up

lw $t6, 0($s6); // t6: left signal(0 or 1)
lw $t7, 0($s7); // t7: right signal(0 or 1)

slt $t0, $s1, $s3;
bne $t0, $zero, Left; // whether barrier drops to the end or not
addi $s1, $zero, 0; // reset to 0
lw $t5, 0($s5); // when reset, get the new random number

Left:
addi $t4, $zero, 1;
bne $t6, $t4, Right;
sub $s0, $s0, $t4;
slt $t4, $s0, $zero;
beq $t4, $zero, Game;
addi $s0, $zero, 79; // deal with left_signal
j Game;

Right:
addi $t4, $zero, 1;
bne $t7, $t4, Game;
addi $s0, $s0, 1;
slt $t4, $s0, $s2;
bne $t4, $zero, Game;
and $s0, $zero, $zero; // deal with right_signal

Game:
lui $t0, 0x000C;
addi $t0, $t0, 3999; // row 51
add $t0, $t0, $s0; // col s0
```

```
addi $t1, $zero, 0xFFF;
sw $t1, 0($t0);
sw $t1, 1($t0);
addi $t0, $t0, 80;
sw $t1, 0($t0);
sw $t1, 1($t0); // draw cube 2*2

or $t2, $s1, $zero;
and $t0, $t0, $zero;
lui $t0, 0x000C;
addi $t1, $zero, 1;
Loop2:
beq $t2, $zero, Next1;
addi $t0, $t0, 80;
sub $t2, $t2, $t1;
j Loop2; // t0 is the first address of the barrier
and $t2, $t2, $zero;
Next1:
addi $t1, $zero, 0xF00;
slt $t3, $t2, $t5;
bne $t3, $zero, draw; // draw the left part
addi $t4, $t5, 10;
slt $t3, $t4, $t2;
bne $t3, $zero, draw; // draw the right part
j No_draw; // can get through
draw:
sw $t1, 0($t0);
sw $t1, 80($t0); // draw
No_draw:
addi $t0, $t0, 1;
addi $t2, $t2, 1;
addi $t4, $s2, 1
beq $t2, $t4, Next2; // whether already drawn the whole line

j Next1;

Next2:
addi $t4, $zero, 51;
bne $s1, $t4, Next3;
slt $t3, $s0, $t5;
bne $t3, $zero, Initial; // hit the barrier
addi $t4, $t5, 8;
slt $t3, $t4, $s0;
bne $t3, $zero, Initial; // hit the barrier
```

```
addi $s4, $s4, 1;
lui $t3, 0xE000;
sw $s4, 0($t3); // store score to GPIO
Next3:
addi $s1, $s1, 1; // barrier goes down for 1 block

addi $t4, $zero, 0;
addi $t3, $zero, 30000;
Delay2:
addi $t4, $t4, 1;
bne $t4, $t3, Delay2; // delay for 30000 times
j Start;
```

第3章 设计实现

3.1 实现方法

（内容要点：实现的具体思路和方法、重点难点在实现中的经验和方法等）

1. 首先将 VGA 模块连接进入顶层，尝试在不经过原有的模块下可否在 VGA 中显示图像，如果显示了预期图像，则说明 VGA 模块基本调通
2. 尝试将 VGA 模块与 CPU 相连，编写简单的汇编程序调试，若符合预期显示，则 VGA 模块与 CPU 模块调通，并画出第一个物体方块
3. 加入按键判断，使用按键控制移位，下板验证
4. 给按键移位加约束，下板验证
5. 尝试画出第二个物体横杠，看看多物体模式下现有代码是否还能兼容，并加入随机数，随机画出缝隙位置
6. 加入设计的判断碰撞的代码，下板验证
7. 完善游戏，下板验证

3.2 实现过程

在原实验 12 的基础上，模块的层次结构图如图:

新增的模块有:vga_controller、VRAM、PS2

重新实现的模块有:MIO_BUS、clk_div; 微调了顶层 Framework 新增的有关信号。

定义地址:

VGA 的内存地址：0000Cxxx

七段数码管的内存地址：E000xxxx

存储器：00000xxx

PS2：000D000x

利用 lw、sw 进行 IO 交互，从以上地址 load 相关的信号，并且通过游戏逻辑进行坐标的运算，并且在汇编代码中完成对 VRAM 的写入。

3.3 仿真与调试

由于本次课程设计包含软核，所以无法进行仿真测试，改为直接下板进行测试。

相关汇编代码进行模拟器单步调试。

在第一步链接 VGA 模块的时候，VGA 一直显示 out of range，一直找不到原因，最后 经过多次分析代码无果，尝试着分析时序问题，查阅笔记后发现时钟周期不对应，更 改后显示出了目标图像。

在调试初期，还可以将许多接口接入七段码管查看数据达到调试的效果。

而 PS2 实现的时候出现了没有反应的情况，经检查是分配的地址与其他地址重合，因此出现冲突，这个 bug 找了很久才发现，可以说不仔细会浪费很多时间。

而在键盘有反应，VGA 有显示之后出现的一系列问题，则通过单步调试汇编解决。

第4章 系统测试验证与结果分析

4.1 功能测试

根据多次试玩，游戏所预期的基本功能:横杠掉落，控制移动，碰撞检测、游戏判断、计算分数都一一实现，都可以在附件中的游戏演示视频中看到具体的实现效果。最终，该设计成为了一个可玩的游戏。

4.2 结果分析

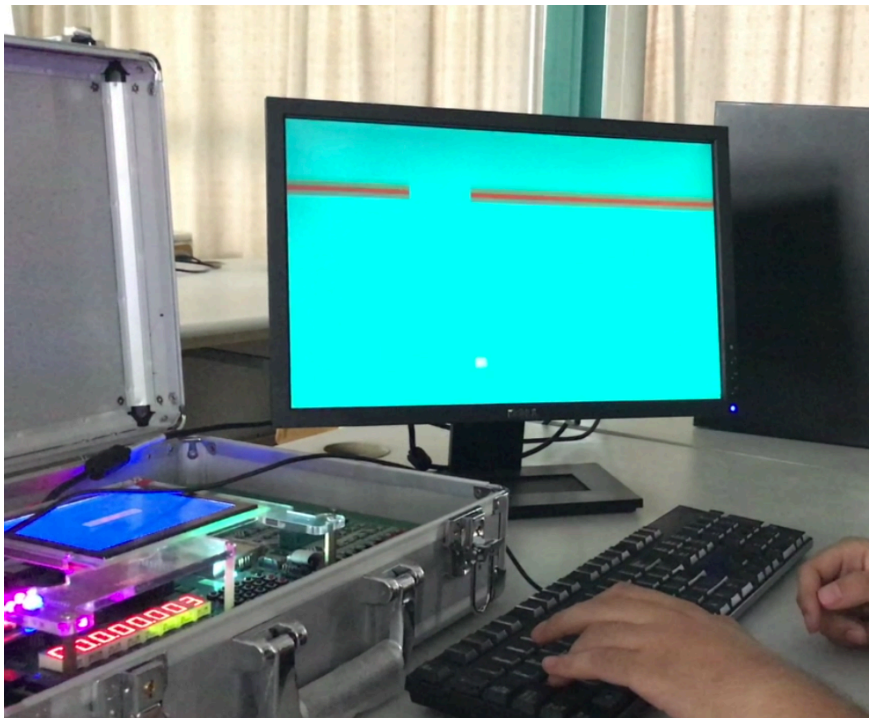
游戏的基本功能均已实现，游戏可以正常的开始与结束，操作过程不会有突变的情况 发生。存在的问题是会有一定程度的闪屏，这是必然存在的，改进方法的思考见第五章的展望。

4.3 系统演示与操作说明

演示主要请看视频，此处列出使用方法，其实非常简单，就是 AD 键控制左右移动穿过障碍，若失败则立刻从头开始，否则积分+1。

如图，可以看到游戏界面和计分板。

PS.需要注意的是，本游戏节奏较快，请助教在操作时迅速移动！



第5章 结论与展望

结论

这次 Project 让我提升很多关于硬件的知识，不同于数逻课程设计的随意搭建模块，这次基于 Framework 的大程更让我有了对计算机组成的系统理解，产生了质的区别。

在做课程设计的过程中发现了许多问题，许多是和其他兴趣小组成员或者同学一起讨论解决的。比如整个流程的理解，VRAM 的使用，汇编的种种 bug 等等。解决过程无非是将

相关的信号线输出到七段码管查看是否异常，分析讨论各种时序问题，找到汇编模拟器进行单步调试等。每一步都至关重要。

感想和建议：我觉得相比其他老师更高权重的考试和无课程设计的设置，施青松老师的课程设计作业非常合理，做的过程中既复习了知识，又能有大作业完成的成就感，还能有另一个机会在平时获得更好的成绩。建议是更详细地讲一讲 MIO_BUS 和显存的原理会更好，刚开始花了一些时间才弄懂，不过这个过程也是十分有价值的。

展望：

在本次课程设计的基础上，我认为还可以进行改进的地方有很多，因为期末时间的原因没有实现，但这些没有实现的部分都不是新的没有被使用的技术，本工程已经达到了所有的基本要求和扩展要求，这些改进是锦上添花。

1. 可以增加开始界面，这可以通过预先储存在 ram 中的汇编指令实现，利用一系列的 lw、sw 可以实现开始界面。
2. 在制作的过程中发现因为刷新较快，每一帧停留时间较短，所以会出现闪屏的现象，这可以通过改进汇编指令，在更新屏幕操作时不进行完全清屏为背景色，而是擦除之前位置的几个小块，这样可以大大减小闪屏的现象，但是汇编指令的复杂度也大大提高，可维护性下降，也可以说是各有优劣。
3. 随机数的生成算法可以改进，但暂时没有更好的想法。