浙江大学

课程设计报告

中文题目 ： 基于 Multi CPU 的 White Tile 游戏

英文题目： WhiteTile Game Design based on MCPU

姓名/学号： 3170103551

指导教师： 施青松

参加成员： 王玥

专业类别： 计算机

所在学院： 计算机学院

论文提交日期 2019 年 6 月 25 日

摘要

计算机组成的大作业，利用 Mips 指令下的多周期框架，进行 White Tile 游戏的设计。自行搭建顶层框架，利用 verilog 语言和 asm 设计，将 asm 转化为 coe 文件加载在 CPU 中。

基本实现游戏功能，添加 VGA 现实和 PS2 键盘交互，利用所学知识，完成整体设计。

关键词： 多周期 CPU Mips Assemble

目录

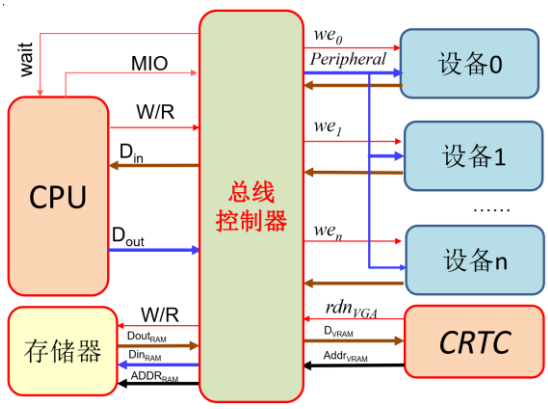
图目录

第1章 绪论

第2章 设计原理

2.1 设计相关内容

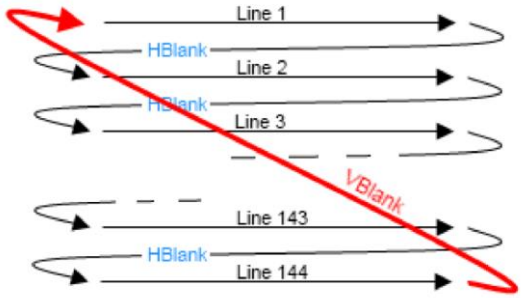
2.1.1 总线控制框架



图表 1 总线框架

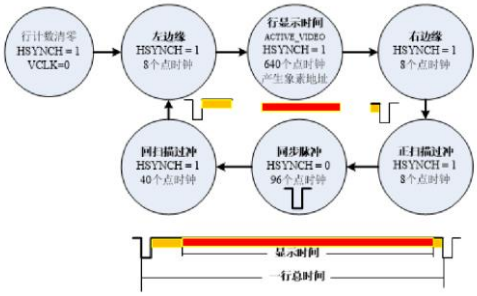
整体 SOC 框架沿用实验框架，其中 MIO_BUS 总线控制框架作为 CPU 和各个其他模块关联的重要部件，在对应的地址线下实现 CPU 控制数据的转换和对不同设备的控制。

2.1.2 VGA 显示原理



图表 2 显示器扫描模式

扫描从屏幕左上角开始，从左向右逐点扫描。 每扫描完一行，电子束回到下一行的起始位置，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，当扫描完全部的行，形成一帧，用场同步信号进行同步，回到屏幕左上角，同时进行场消隐，开始下一帧



图表 3 扫描机状态

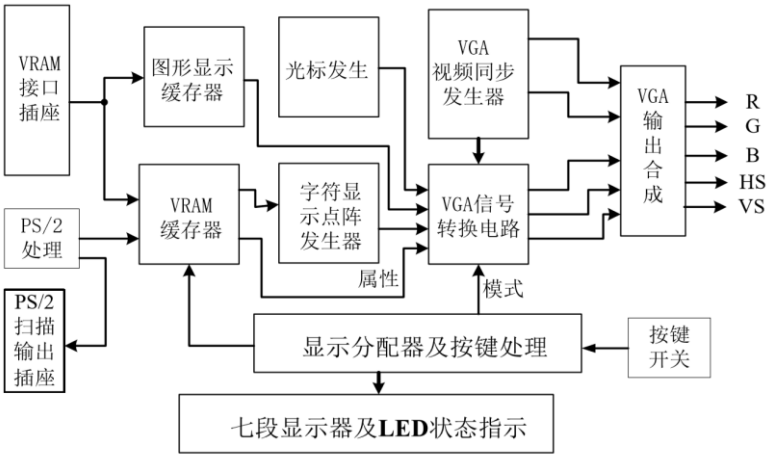
同步信号每个周期可以分为四个时间段：

Tdisp：显示时间

Tpw：负脉冲（消隐）时间

Tfp：前沿时间（消隐信号的前沿）

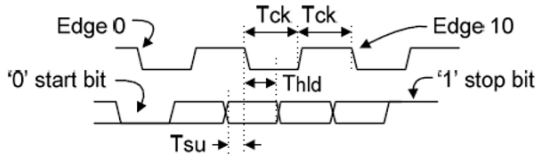
Tbp：后沿时间（消隐信号的后沿）



图表 4 显示接口

显示时，通过场同步和行同步实现同步信号输出，颜色信号分为 12 位，RGB 三个颜色通道各 4 位。

2.1.3 PS2 键盘设计



图表 5 PS2 键盘传输时序

单次传送数据控制位有：

ps2_clk,ps2_data:空闲时高电平，传输数据时产生信号

start:当 data 在 clk 的下降沿从高电平变为低电平

parity:用于奇偶校验以降低误码率

stop:单次传输数据结束位

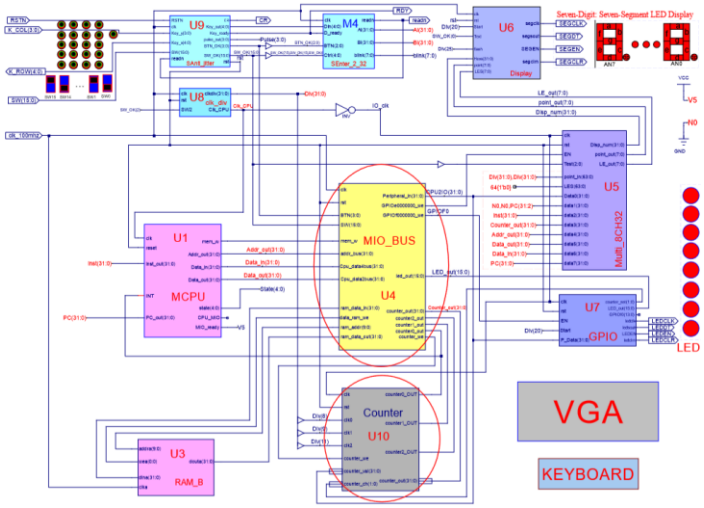
当键盘发送数据时，时钟信号的下降沿被读取；

当主机发送数据时，时钟信号的上升沿被读取。

2.2 设计方案

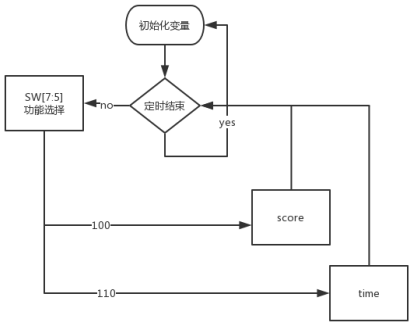
2.2.1 整体设计方案

结构框图：



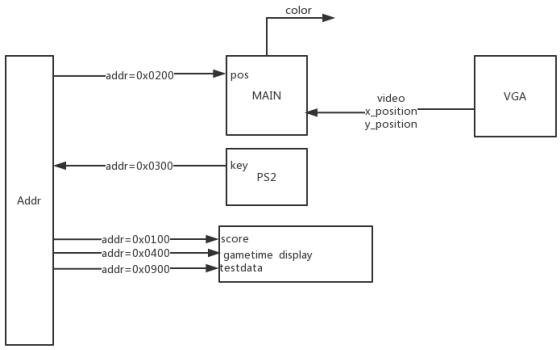
图表 6 顶层框架

系统流程图：



图表 7 系统流程图

主要接口逻辑：



图表 8 主要接口逻辑

2.3 硬件设计

这里主要列出和课程实验内容不一样或者有更改的地方。

2.3.1 顶层模块

顶层模块基本和实验 13 一致，其中增加了 VGA、PS2 和 TOP 模块，TOP 模块负责解码 CPU 传出的 position 数据到 VGA 的 color 上,VGA 负责输出板子的扫描信号,PS2 输出键盘的输入代码

其中主要模块的接口关系见图表 8，主要代码如下：

```
module Multi_SOC(input RSTN,
            output [4:0]BTN_x,
            input[3:0]BTN_y,
            input [15:0]SW,
            input clk_100mhz,
            output CR,
            output RDY,
            output readn,
            output seg_clk,
            output seg_sout,
            output SEG_PEN,
            output seg_clrn,
            output led_clk,
            output led_sout,
            output LED_PEN,
            output led_clrn,
            output [7:0]SEGMENT,
            output [3:0]AN,
            output [7:0]LED,

            input PS2_clk,
            input PS2_data,
            output [3:0]Red,
            output [3:0]Green,
            output [3:0]Blue,
            output h_sync,
            output v_sync
    );
wire V5,N0;
assign V5=1'b1;
assign N0=1'b0;

wire [4:0]Key_out;
wire [3:0]Pulse;
wire [15:0]SW_OK;
wire [3:0]BTN_OK;
```

```
37
38  wire [63:0]point_out_U5,LES;
39  wire [31:0]Data1;
40  assign point_out_U5={Div[31:0],Div[31:13],State[4:0],NO,NO,NO,NO,NO,NO,NO,NO};
41  assign Data1={NO,NO,PC[31:2]};
42  assign LES={64'b0000000000000000000000000000000000000000000000000000000000000000};
43  wire [31:0]test_addr,test_pos;
44  wire [31:0]test_CPUpos;
45  assign test_CPUpos={20'h0,CPUpos};
46  SAnti_jitter U9(.RSTN(RSTN),
47                                                      .clk(clk_100mhz),
48                                                      .Key_y(BTN_y),
49                                                      .Key_x(BTN_x),
50                                                      .SW(SW),
51                                                      .readn(readn),
52                                                      .CR(CR),
53                                                      .Key_out(Key_out),
54                                                      .Key_ready(RDY),
55                                                      .pulse_out(Pulse),
56                                                      .BTN_OK(BTN_OK),
57                                                      .SW_OK(SW_OK),
58                                                      .rst(rst)
59                                                     );
60  wire [31:0]Div;
61  wire IO_clk,Clk_CPU;
62  assign IO_clk = ~Clk_CPU;
63  clk_div U8(.clk(clk_100mhz),
64                                      .rst(rst),
65                                      .SW2(SW_OK[2]),
66                                      .clkdiv(Div),
67                                      .Clk_CPU(Clk_CPU)
68                                     );
69
70  wire [31:0]Ai,Bi;
71  wire [7:0]blink;
72  wire [4:0]Ctrl;
73  assign Ctrl = {SW_OK[7:5],SW_OK[15],SW_OK[0]};
74  SEnter_2_32 M4(.clk(clk_100mhz),
75                                                      .Din(Key_out),
76                                                      .D_ready(RDY),
77                                                      .BTN(BTN_OK[2:0]),
78                                                      .Ctrl(Ctrl),
79                                                      .readn(readn),
80                                                      .Ai(Ai),
```

```
                                                    .Bi(Bi),
                                                    .blink(blink)
                                                    );
wire [7:0]LE_out,point_out;
wire [31:0]Disp_num;
SSeg7_Dev U6(.clk(clk_100mhz),
                                                .rst(rst),
                                                .Start(Div[20]),
                                                .SW0(SW_OK[0]),
                                                .flash(Div[25]),
                                                .Hexs(Disp_num),
                                                .point(point_out),
                                                .LES(LE_out),
                                                .seg_clk(seg_clk),
                                                .seg_sout(seg_sout),
                                                .SEG_PEN(SEG_PEN),
                                                .seg_clrn(seg_clrn)
                                               );
wire [1:0]counter_set;
wire [15:0]LED_out;
wire [31:0]CPU2IO;//out from MIO_BUS
wire GPIOF0;//out from MIO_BUS
SPIO U7(.clk(IO_clk),
                          .rst(rst),
                          .EN(GPIOF0),
                          .Start(Div[20]),
                          .P_Data(CPU2IO),

                          .counter_set(counter_set),
                          .LED_out(LED_out),
                          .GPIOf0(),
                          .led_clk(led_clk),
                          .led_sout(led_sout),
                          .LED_PEN(LED_PEN),
                          .led_clrn(led_clrn)
                          );
wire counter0_OUT,counter1_OUT,counter2_OUT;
wire [31:0]counter_out;
wire        counter_we;//out from MIO_BUS
Counter_x U10(.clk(IO_clk),
                                                .rst(rst),
                                                .clk0(Div[8]),
                                                .clk1(Div[9]),
                                                .clk2(Div[11]),
```

```
                                                .counter_we(counter_we),
                                                .counter_ch(counter_set),
                                                .counter_val(CPU2IO),

                                                .counter0_OUT(counter0_OUT),
                                                .counter1_OUT(counter1_OUT),
                                                .counter2_OUT(counter2_OUT),
                                                .counter_out(counter_out)
                                                );


wire mem_w;
wire [31:0]Addr_out,Data_out,inst,PC;
wire [4:0]State;
wire [31:0]Data_in;//out from MIO_BUS
Multi_CPU U1(.clk(Clk_CPU),
                                                .reset(rst),
                                                .INT(counter0_OUT),
                                                .MIO_ready(V5),
                                                .Data_in(Data_in),

                                                .mem_w(mem_w),
                                                .Addr_out(Addr_out),
                                                .Data_out(Data_out),
                                                .state(State),
                                                .CPU_MIO(),
                                                .inst_out(inst),
                                                .PC_out(PC)
                                                );
wire [31:0]ram_data_out;
wire [9:0]ram_addr;//out from MIO_BUS
wire [31:0]ram_data_in;
wire data_ram_we;//out from MIO_BUS
RAM_B U3(.clka(clk_100mhz),
                                    .addra(ram_addr),
                                    .wea(data_ram_we),
                                    .dina(ram_data_in),

                                    .douta(ram_data_out)
                                    );
wire GPIOE0;
wire [2:0]Scan;
assign Scan={SW_OK[1],Div[19:18]};
Seg7_Dev U61(.Scan(Scan),
```

```
                                                .SW0(SW_OK[0]),
                                                .flash(Div[25]),
                                                .Hexs(Disp_num),
                                                .point(point_out),
                                                .LES(LE_out),

                                                .SEGMENT(SEGMENT),
                                                .AN(AN)
                                                );
PIO U71(.clk(IO_clk),
                            .rst(rst),
                            .EN(GPIOF0),
                            .PData_in(CPU2IO),

                            .counter_set(),
                            .LED_out(LED),
                            .GPIOf0()
                            );

//----------for vga and ps2 and key
wire [15:0]key;
wire [11:0]CPUpos;
wire [31:0]testdata,score;
wire clk_25mhz,inside_video;
wire [9:0]x_position;
wire [8:0]y_position;
assign clk_25mhz=Div[1];
vga_controller VGA (
                            .clk(clk_25mhz),
                            .reset(rst),
                            .h_sync(h_sync),
                            .v_sync(v_sync),
                            .inside_video(inside_video),
                            .x_position(x_position),
                            .y_position(y_position)
            );
wire [11:0] color;
assign Red = inside_video ? color[11:8] : 4'b0000;
assign Green = inside_video ? color[7:4] : 4'b0000;
assign Blue = inside_video ? color[3:0]: 4'b0000;
ps2 keyboard(.clk(Div[0]),.reset(rst),.PS2_clk(PS2_clk),.PS2_data(PS2_data),.key(key));
wire [31:0]gametime;

MAIN top(.clk(clk_25mhz),
```

```
                                    .video(inside_video),
                                    .datafromCPU(CPUpos),
                                    .x(x_position),
                                    .y(y_position),
                                    .color(color),
                                    .test_addr(test_addr),
                                    .test_pos(test_pos)
                                    );

Multi_8CH32 U5(.clk(IO_clk),
                                                        .rst(rst),
                                                        .EN(GPIOE0),
                                                        .Test(SW_OK[7:5]),
                                                        .point_in(point_out_U5),
                                                        .LES(LES),
                                                        .Data0(CPU2IO),
                                                        .data1(test_CPUpos),
                                                        .data2(test_pos),
                                                        .data3({key,key}),
                                                        .data4(score),
                                                        .data5(testdata),
                                                        .data6(gametime),
                                                        .data7(PC),

                                                        .Disp_num(Disp_num),
                                                        .point_out(point_out),
                                                        .LE_out(LE_out)
                                                        );
MIO_BUS U4(.clk(clk_100mhz),
                                    .rst(rst),
                                    .BTN(BTN_OK),
                                    .SW(SW_OK),
                                    .mem_w(mem_w),
                                    .led_out(LED_out),
                                    .addr_bus(Addr_out),
                                    .Cpu_data2bus(Data_out),
                                    .ram_data_out(ram_data_out),
                                    .counter_out(counter_out),
                                    .counter0_out(counter0_OUT),
                                    .counter1_out(counter1_OUT),
                                    .counter2_out(counter2_OUT),

                                    .Cpu_data4bus(Data_in),
                                    .ram_data_in(ram_data_in),
```

```
257                                     .data_ram_we(data_ram_we),
258                                     .ram_addr(ram_addr),
259                                     .GPIOf0000000_we(GPIOF0),
260                                     .GPIOe0000000_we(GPIOE0),
261                                     .counter_we(counter_we),
262                                     .Peripheral_in(CPU2IO),
263
264                                     .video(inside_video),
265                                     .key(key),
266                                     .pos(CPUpos),
267                                     .score(score),
268                                     .testdata(testdata),
269                                     .gametime(gametime)
270                                     );
271 endmodule
```

2.3.2 VGA 模块

VGA 模块通过扫描信号输出此时 x，y 的点位置和行、场同步信号，主要分成行同步信号计时、hsync 的脉冲信号产生两大部分。VGA 产生的信号传送给 MAIN 进行判断，从而得到 color 值输出到显示屏，扫描时钟为 25mhz。

具体原理解析见原理部分，主要代码如下：

```
module vga_controller(input clk,        // clk_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 clk 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 clk 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
```

图表 9 VGA 代码

2.3.3 PS2 模块

PS2 通过 clk 和 data 数据得到键盘所得的 key 编码值,并传送给 CPU 进行处理,扫描时钟为 100mhz。

具体 PS2 代码如下:

```
module ps2(input clk,   //25mhz
                          input reset,
                          input PS2_clk,
                          input PS2_data,
                          output reg[15:0]key
    );
        reg PS2_clkf, PS2_dataf;
        reg [7:0] PS2_clk_filter, PS2_data_filter;
        reg [10:0] shift1, shift2;

        wire [15:0]xkey;
        assign xkey = {shift2[8:1], shift1[8:1]};

        // Filter for PS2 clock and data
        always @ *
        begin
                key = xkey;
        end

        always @ (posedge clk or posedge reset)
        begin
                if (reset == 1)
                begin
                        PS2_clk_filter <= 0;
                        PS2_data_filter <= 0;
                        PS2_clkf       <= 1;
                        PS2_dataf       <= 1;
                end
                else
                begin
                        PS2_clk_filter <= {PS2_clk, PS2_clk_filter[7:1]};
                        PS2_data_filter <= {PS2_data, PS2_data_filter[7:1]};
                        if (PS2_clk_filter == 8'b1111_1111)
                                PS2_clkf <= 1;
                        else if (PS2_clk_filter == 8'b0000_0000)
                                PS2_clkf <= 0;
                        if (PS2_data_filter == 8'b1111_1111)
                                PS2_dataf <= 1;
                        else if (PS2_data_filter == 8'b0000_0000)
                                PS2_dataf <= 0;
                end
        end

        // Shift register used to clock in scan codes from PS2
        always @ (negedge PS2_clkf or posedge reset)
        begin
                if (reset == 1)
                begin
                        shift1 <= 0;
                        shift2 <= 1;
                end
                else
                begin
                        shift1 <= {PS2_dataf, shift1[10:1]};
                        shift2 <= {shift1[0], shift2[10:1]};
                end
        end

endmodule
```

图表 10 PS2 代码

2.3.4 MIO_BUS 模块

MIO_BUS 为总线控制模块，控制 CPU 数据的输入和输出，其中主要涉及到的为 key、position、score、gametime 和 testdata 几个接口，具体总线控制地址分配和接口见图表 8.主要 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,                                            //addr fron CPU
                                    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[9:0]ram_addr,                                                //Memory Address signals
                                    output reg data_ram_we,
                                    output reg GPIOf0000000_we,     //GPIOffffff00_we
                                    output reg GPIOe0000000_we,     //GPIOffffffe00_we
                                    output reg counter_we,                                          //计数器
                                    output reg[31:0]Peripheral_in,                  //连接外部的总线

                                    input [31:0] lg_out,
                                    output reg lg_we,
                                    output reg [6:0] lg_addr,
                                    // input wire [31:0] score,
                                    // output reg [31:0] score_reg,
                                    // output reg score_reg_we,
                                    input video,
                                    input [15:0]key, // keycode
                                    output reg[11:0]pos, // the pos of tile
                                    output reg[31:0]score, // the score
                                    output reg[31:0]testdata, // testprocess
                                    output reg[31:0]gametime
                                    );
 reg data_ram_rd,GPIOf0000000_rd,GPIOe0000000_rd,counter_rd;
 reg [7:0] led_in;
 reg lg_rd;
 initial begin
        score = 32'h0;
        pos = 32'h06b;
        testdata = 32'hFFFFFFFF;
 end
//RAM & IO decode signals
always @* begin
       data_ram_we = 0;        //主存写信号
       data_ram_rd = 0;        //主存读信号
       counter_we = 0; //计数器写信号
       counter_rd = 0; //计数器读信号
       GPIOf0000000_we = 0;    //设备1：PIO写信号
       GPIOe0000000_we = 0;    //计数器：Counter_x写最好
       GPIOf0000000_rd = 0;    //设备3、4：SW等读信号
       GPIOe0000000_rd = 0;    //设备2：七段显示器写信号
       ram_addr = 10'h0;       //内存数据地址：RAM--B地址
       ram_data_in = 32'h0;    //内存读数据：RAM B输出数据
       Peripheral_in = 32'h0;  //外设总线：CPU输出，外设写入数据
       Cpu_data4bus = 32'h0;   //开始译码 //data_ram(00000000 - 00000ffc,actually lower 4KB RAM)

       lg_we = 0;
       lg_rd = 0;
       lg_addr = 7'b0;

       case(addr_bus[31:28])
       4'h0:begin
                        if(addr_bus == 32'h00000100) begin   //s0 is addr of score 0x0100
                                if(mem_w)
                                        score = Cpu_data2bus;
                                else
                                        Cpu_data4bus = score;
                        end

                        else if (addr_bus == 32'h00000200) begin // position 0x0200
                                if(mem_w && video)
                                        pos = Cpu_data2bus[11:0];
                                else
                                        Cpu_data4bus = {20'h0,pos};
                        end

                        else if (addr_bus == 32'h00000300) begin // ps2 addr 0x0300
                                //Peripheral_in=Cpu_data2bus;
                                Cpu_data4bus = {16'h0000,key}; // send ps2 to cpu
                        end

                        else if (addr_bus == 32'h00000400) begin //gametime addr 0x400
                                if(mem_w)
                                        gametime = Cpu_data2bus;
                                else
                                        Cpu_data4bus = gametime;
                        end

                        else if (addr_bus == 32'h00000900) begin //testdata addr 0x900
                                if(mem_w)
                                        testdata = Cpu_data2bus;
                                else
                                        Cpu_data4bus = testdata;
                        end

                        else 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
                end

       4'he:begin       //七段显示器 (e0000000 - effffffff , SSeg7_Dev,  GPIO/LED显示地址)
                        GPIOe0000000_we=mem_w;
                        Peripheral_in=Cpu_data2bus;
                        Cpu_data4bus = counter_out;
                        GPIOe0000000_rd=~mem_w;
                end

       4'hf:begin       //PIO (f0000000-fffffff0 , 8LEDs - GPIO/Switch. BTN地址 & conunter, f000004-fffffff4)
                        if(addr_bus[2])begin
                                counter_we=mem_w;
                                Peripheral_in=Cpu_data2bus;     //write Counter Value. f000004-fffffff4
                                Cpu_data4bus = counter_out;
                                counter_rd=~mem_w;
                        end
                        else begin
                                GPIOf0000000_we=mem_w;
                                Peripheral_in=Cpu_data2bus;     //write Counter set & Initialization and light LED. f0000000-fffffff0
                                Cpu_data4bus = {counter0_out, counter1_out, counter2_out, 9'h00, led_out, BTN, SW};
                                GPIOf0000000_rd=~mem_w;
                        end
                end
       endcase

       casex ({data_ram_rd, lg_rd, GPIOe0000000_rd, counter_rd, GPIOf0000000_rd})
                        5'b1xxxx: Cpu_data4bus = ram_data_out;  // read from RAM
                        5'bx1xxx: Cpu_data4bus = lg_out;        // read from Life game
                        5'bxx1xx: Cpu_data4bus = counter_out;   // read from Counter
                        5'bxxx1x: Cpu_data4bus = counter_out;   // read from Counter
                        5'bxxxx1: Cpu_data4bus = {counter0_out, counter1_out,  counter2_out, 9'h00, led_out, BTN, SW};  //read from SW & BTN
       endcase

end

endmodule
```

图表 11 MIO_BUS

2.3.5 MAIN 模块

MAIN 模块通过 VGA 和 position 的数据来判断 color 的输出。

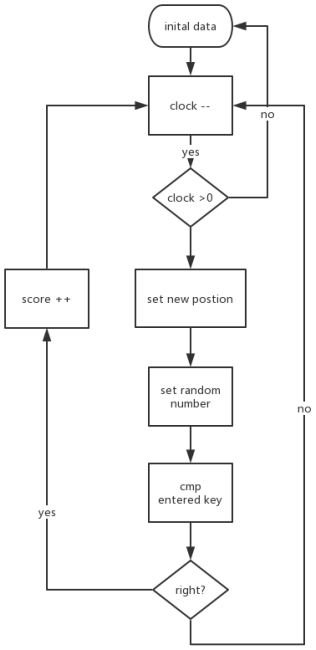
MAIN 代码：

```
module MAIN(input clk,
                                        input [11:0]datafromCPU,
                                        input [9:0]x,
                                        input [8:0]y,
                                        input video,
                                        output reg[11:0] color,
                                        output [31:0]test_pos,
                                        output [31:0]test_addr
    );
            wire [11:0]pos;
            assign pos=datafromCPU[11:0];
            wire [1:0]blockx;
            assign blockx=(160 <= x) + (320 <= x) + (480 <= x);
            wire [1:0]liney;
            assign liney=(y <= 160 ) + (y <= 340);
            wire [3:0]addr;
            assign addr=liney * 4 + blockx; //现在的 vga 位置
            always @ (posedge clk)begin
                        if(video)begin
                                    if((addr == pos[11:8] ) | (addr == pos[7:4] ) | (addr == pos[3:0] ))
                                                color <= 12'h000;
                                    else
                                                color <= 12'hFFF;
                        end
            end

            assign test_pos = {20'h0,pos};
            assign test_addr = {blockx,28'h0,liney};
endmodule
```

2.4 系统软件设计

2.4.1 ASM 设计流程图



图表 12 ASM 设计流程图

2.4.2 ASM 代码设计

ASM 代码主要时控制 RAM 实现 CPU 的数据交互和操作,本游戏中是主要代码模块。下面将对几个重要的模块设计实现过程进行介绍。

2.4.2.1 setposition

用来生成下一个时钟的 position 位置，这里的 postion 指的是 VGA 中黑白模块的编码结果，具体编码方式见测试模块的表格。

setposition 分成两个部分，第一个部分是对上一次的 position 的首位位置去除并将后两个位置前移动，由编码可知每一层只有一个黑色方块，黑色方块的位置由一个 16 进制表示，也就是 4bit 数据，设三层的表示方式为{x1，x2，x3}.

一个时钟结束后，按照编码方式，我们要将原来的{x1,x2,x3}转变为{x2,x3,x1`}，其中实现前面两位前移动，通过 and 0x00FF 清除第一位，再整体左移动 4 位也就是两两相加两次。这里注意，左移后还要分别减去 4，实现编码意义上的层数下降。

实现新的方块位置实现，我们通过剩余时间模拟伪随机数生成的方式，由于总过需要的循环数字只有四个，伪随机数生成的方式也非常简单。直接用时间种子和此时时间相加后得出以后 mod3，也就是 and 0x0003.得到了新一轮的黑色方块位置，还需要增加 0x0008 实现编码位置是最上面一层。

2.4.2.2 cmpkey

cmpkey 需要先加载 key 的值，通过 PS2 读取 key 值，后比较 key 的值。得到 key 具体是 1、2、3、4 中的位置以后和当前的 position 位置比较，比较 position 的 x1 和 0、1、2、3 的关系，通过 and 来实现，如果相同则跳转到 right 模块，否则位 wrong 模块。

2.4.2.3 具体完整代码

```
j start
start: // ------------------start----------------
addi $s0, $zero, 0x0100 // s0 == addr(score),0x0100
addi $s1, $zero, 0x0200 // s1 == addr(position),0x0200
addi $s2, $zero, 0x0300 // s2 == addr(ps2),0x0300
addi $s3, $zero, 0x0400 // s3 == addr(gametime),0x0400
addi $s4, $zero, 0x0900 // s4 ==  addr(testdata),0x900
j initial
initial: // ----------------initial-----------
addi $t3,$zero,0x003c //initial time=0x003c=60
addi $t1,$zero,0x006B //initial position=0x006B
addi $t0,$zero,0x0000 //initial score=0x0000
sw $t0,0($s0) //store score
sw $t1,0($s1) //store position
j cmpkey

setposition: //----------setposition--------
j subpos1
setrandom: //----------setrandom------
addi $t3,$t3,-0x0001 // clock--
sw $t3,0($s3) // store gametime
beq $t3,$zero,initial
add $t2,$t2,$t3 //t2=t2+clock
add $t6,$zero,$zero
addi $t6,$t6,0x0003
and $t6,$t6,$t2
add $t1,$t1,$t6
addi $t1,$t1,0x0008
sw $t1,0($s1)
j cmpkey

subpos1: //--------------subpos1-------
add $t6,$zero,$zero
addi $t6,$t6,0x00FF
and $t1,$t1,$t6
add $t1,$t1,$t1 // t1<<1
add $t1,$t1,$t1 // t1<<1
```

```
38  add $t1,$t1,$t1 // t1<<1
39  add $t1,$t1,$t1 // t1<<1
40  addi $t1,$t1,-0x0440
41  j setrandom
42
43
44  cmpkey: //--------------cmpkey----------
45  lw $t7,0($s2) //t7 = ps2
46  lw $t7,0($s2) //t7 = ps2
47  lw $t7,0($s2) //t7 = ps2
48  lw $t7,0($s2) //t7 = ps2
49  addi $t6,$zero,0x00FF
50  and $t7,$t7,$t6
51  sw $t7,0($s4) //store testdata
52  addi $t6,$t7,-0x0016 //t6=key - 1616(1)
53  beq $t6, $zero, key1
54  addi $t6,$t7,-0x001E //t6=key - 1E1E(2)
55  beq $t6, $zero, key2
56  addi $t6,$t7,-0x0026 //t6=key - 2626(3)
57  beq $t6, $zero, key3
58  addi $t6,$t7,-0x0025 //t6=key - 2525(4)
59  beq $t6, $zero, key4
60  j setposition
61
62  key1:  // ----------------------------------------------key1----------------------
63  addi $t6,$t6,0x0100
64  slt $t6,$t1,$t6
65  beq $t6, $zero, wrong
66  j right
67  key2:  // ----------------------------------------------key2----------------------
68  addi $t6,$t6,0x0100
69  and $t6,$t1,$t6
70  slt $t6,$zero,$t6
71  beq $t6, $zero, wrong
72  j right
73  key3:  // ----------------------------------------------key3----------------------
74  addi $t6,$t6,0x0200
75  and $t6,$t1,$t6
76  slt $t6,$zero,$t6
77  beq $t6, $zero, wrong
78  j right
79  key4:  // ----------------------------------------------key4----------------------
80  addi $t6,$t6,0x0300
81  and $t6,$t1,$t6
```

```
slt $t6,$zero,$t6
beq $t6, $zero, wrong
j right

wrong:  // ----------------------------------------wrong----------------------
j setposition
right:  // ----------------------------------------right----------------------
addi $t0,$t0,0x0001 // score ++
sw $t0,0($s0)       // store score
j setposition
```

2.4.3 COE 转化

COE 转化即通过汇编器将 ASM 文件转化成 RAM 可执行的 COE 文件，其中设置 width 为 32，depth 为 1024，生成新的 RAM 核

通过张海学长的汇编器实现 COE 转化，COE 内容如下：

```
memory_initialization_radix=16;
memory_initialization_vector=
08000001, 20100100, 20110200, 20120300, 20130400, 20140900, 08000007, 200B003C,
2009006B, 20080000, AE080000, AE290000, 08000022, 08000019, 216BFFFF, AE6B0000,
1160FFF6, 014B5020, 00007020, 21CE0003, 01CA7024, 012E4820, 21290008, AE290000,
08000022, 00007020, 21CE00FF, 012E4824, 01294820, 01294820, 01294820, 01294820,
2129FBC0, 0800000E, 8E4F0000, 8E4F0000, 8E4F0000, 8E4F0000, 200E00FF, 01EE7824,
AE8F0000, 21EEFFEA, 11C00007, 21EEFFE2, 11C00009, 21EEFFDA, 11C0000C, 21EEFFDB,
11C0000F, 0800000D, 21CE0100, 012E702A, 11C00010, 08000046, 21CE0100, 012E7024,
000E702A, 11C0000B, 08000046, 21CE0200, 012E7024, 000E702A, 11C00006, 08000046,
21CE0300, 012E7024, 000E702A, 11C00001, 08000046, 0800000D, 21080001, AE080000,
0800000D;
```

第3章 设计实现

3.1 实现方法

主要实现方法就是通过利用之前实验 13 的 SOC 顶层结构，多周期设计的 CPU，增加 BUS 总线控制方式，结合生成核的 coe 代码实现。额外增加了 PS2 的键盘和 VGA 的显示模块。

3.2 实现过程

3.2.1 实现过程

实现过程：

先利用实验 13 的模块进行 VGA 和 PS2 接口的借入实现，用七段数码管测试 key 编码结果和 VGA 编码结果的正确性。

再利用实验 13 的模块检测 asm 代码的正确性，主要有两个部分：通过时钟生成随机数使得屏幕显示滚动的效果、通过 key 值判断正确性得到得分。

当每一个小部分实现成功后，两个部分结合起来

3.2.2 模块调用层级

模块调用：

图表 13 模块调用层级图

3.3.3 仿真测试

仿真：

由于许多模块直接调用 NGC 核，所以不能整体仿真，故而分成 PS2 的数据读取、VGA 的编码模块和 ASM 数据读取三个部分进行测试。

3.3 仿真与调试

3.3.1 仿真

- ERROR:HDLCompiler:718 - "C:\Users\wangyue\Desktop\Pro\code\Exp13_MSOC\code\Multi_SOC.v" Line 266: Port connections cannot be mixed ordered and named
  ERROR:HDLCompiler:45 - "C:\Users\wangyue\Desktop\Pro\code\Exp13_MSOC\code\Multi_SOC.v" Line 266: PS2_data is not a function
  ERROR:HDLCompiler:24 - "C:\Users\wangyue\Desktop\Pro\code\Exp13_MSOC\code\Multi_SOC.v" Line 266: PS2_data expects 0 arguments
  引用端口的时候格式为(.port(wire))，缺少.则报错误，引用函数可以直接引用。
- Xst:1710 - FF/Latch <state_out_FSM_FFd7> (without init value) has a constant value of 0 in block <ctrl>. This FF/Latch will be trimmed during the optimization process.
  报warning，但是最够generate的结果正确。
- ERROR:HDLCompiler:718 - "C:\Users\wangyue\Desktop\Pro\code\test\code\Multi_SOC.v" Line 279: Port connections cannot be mixed ordered and named
  模块调用最后多了逗号。
- ERROR:HDLCompiler:35 - "C:\Users\wangyue\Desktop\Pro\code\Exp13_MSOC\code\Multi_SOC.v" Line 212: <testdata> is already implicitly declared earlier.
  顶层模块调用时，wire的命名有顺序，在先写出的模块中命名后面就会报错，尽管模块之间没有时序关系。

3.3.2 调试过程

- VGA 调试的格子错乱

  是编码转换的问题，重新调整了编码的方式，把格子巧妙地实现黑色格子的定位和三层每层只有一个黑色格子的结果。

- asm 初始化 PC 值不变

  检查 SW[7:5]=111 的 PC 输出发现，PC 值只在 84，88，9C 之间跳动，CPU 数据输出跳动。猜测可能是产生了延迟的问题。

  所以单独把 coe 文件加载到了实验 13 的 SOC 测试构架上调试，经过了许多时间的调试，检查 cpu 输出，发现最开始的数据没有加载成功。

  再缩小 asm 的测试程序，发现是实验 13 的 ctrl 中的 addi 状态机写的有些问题，修改以后就好了。

- key 的调试

  发现综合起来以后，按键总是很难得到正确的结果，后来发现只有长按键，读取到的 key 值才是 1616，这样短按的时候很难刚好卡在读取的点上，所以就会显得不太对。

  后来扩大判断，F016 这样的断言情况也包括进去，这样只要按键按下的时间在读取 key 值之前就都是正确的，虽然依旧手慢一点还是读不到。所以增加了 lw 的次数来扩大反应时间要求，考虑到手速是游戏要求之一，就没有再增多时间，只给了四个周期的时间。

- score 的延迟

  发现得分一直有延迟，由于判断过程有许多个周期，所以需要时间来处理，最好的方法只有减少处理语言条数和增加时钟速度，尽量取得比较好平衡的次数，语言尽可能写的最精简了。

第4章 系统测试验证与结果分析

4.1 功能测试

4.1.1 图形界面测试

图形界面为简单的黑白格子，初始化位置为 06b，位置如下

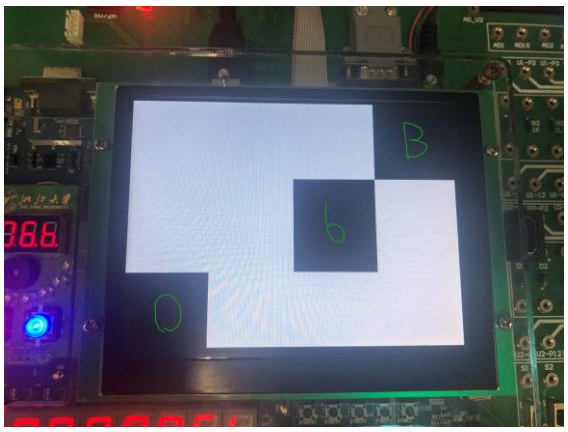


Figure 2 初始化格子情况

初始化接面，3X4 的黑白格子，初始位置被定义为 06b，格子和代码转化过程见测试参数，此处仅列出初始化情况，其他情况见系统测试使用模块。

4.1.2 键盘测试

键盘测试：

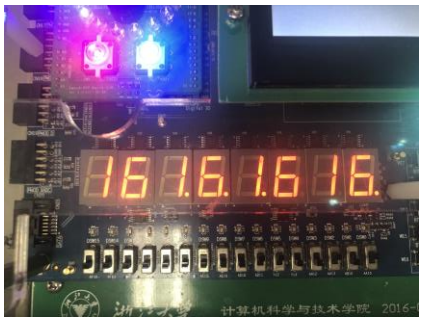


Figure 3 key=1

Figure 4 key=2

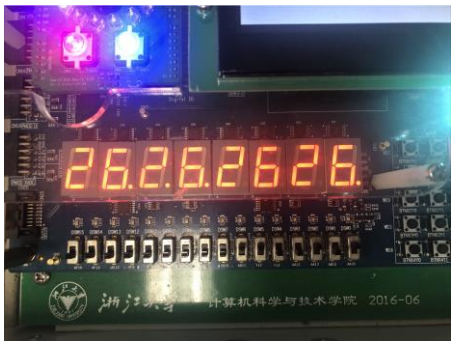


Figure 5 key=3

Figure 6 key=4

七段数码管拨到 SW[7：5]为 011 时显示{key，key}，主要测试为游戏设计所需要的键（1，2，3，4），分别对应的 keycode 为（1616, 1E1E, 2626, 2525）

4.2 技术参数测试

表格 1 测试按键功能

按键参数		
SW[7:5]	值	功能
	000	CPU 取出的数值
	001	CPU 取出 VGA 数据
	010	VGA 扫描输入状态
	011	PS2 读入键盘编码值
	100	score
	101	test 过程量
	110	剩余 gametime 时间
	111	PC 指令地址

表格 2 VGA 格子编码

VGA 格子编码			
1001（9）	1010（A）	1011（B）	1100（C）
0101（5）	0110（6）	0111（7）	1000（8）
0001（1）	0010（2）	0011（3）	0100（4）

表格 3 keycode

keycode 编码	
键盘输入	编码
1	1616
2	1E1E
3	2626
4	2525

表格 4 regs 使用

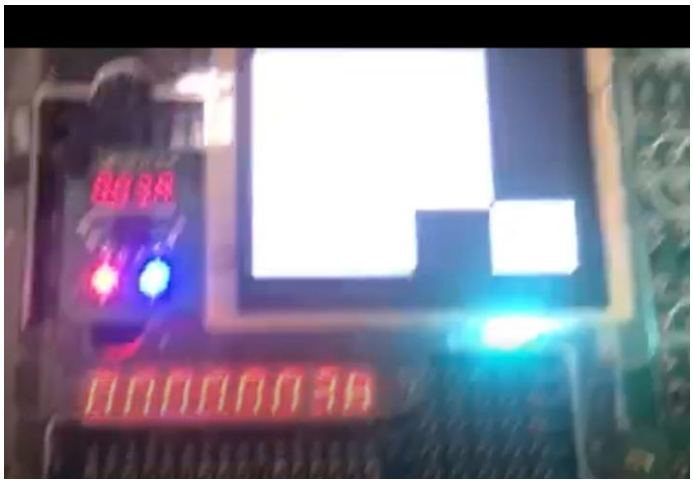
使用寄存器数据		
数据地址	寄存器值	功能
0x0100	$s0	score
0x0200	$s1	position
0x0300	$s2	ps2
0x0400	$s3	gametime
0x0900	$s4	testdata
寄存器	功能	
$t0	score	
$t1	position	
$t3	time	
$t6	判断	
$t7	ps2	

4.3 结果分析

测试所有功能完备，结果正确

4.4 系统演示与操作说明

第一步：初始化第一个图像 06b

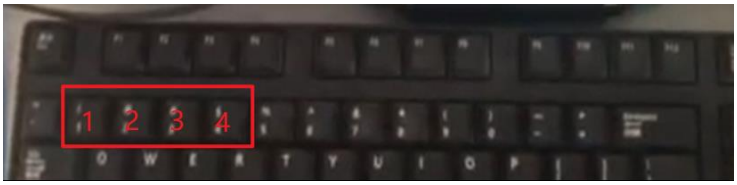


图表 14 初始化图像

第二步：打开 SW[7:5]状态为 100，显示此时分数

图表 15 SW[7:5]=100

第三步：敲击键盘 1、2、3、4 键，跟上显示屏的速度



图表 16 键盘按键

第四步：打开 SW[7:5]状态为 110，显示剩余时间

图表 17 SW[7:5]=110

第五步：调整 SW[2]来调整游戏速度

图表 18 SW[2]调整速度

具体的实现过程测评见视频演示。

第5章 结论与展望

基本完成了游戏设计，其中由于数字逻辑大作业对PS2和VGA有了一定的了解，这一块设计难度不大，更多的时间花费在了游戏主题构架的设计和对SOC整体框架的理解之上。

通过游戏设计，总体花费了将近20个小时时间，更好地理解了多周期CPU的处理过程，在希望完成的功能上都得到了良好的实现，但是过程依旧还有一些小毛病没有解决。由于指令是一步一步进行的，得分的显示和操作有一定的延迟度，对游戏体验不是很好，暂时没有想出比较好的解决方案。键盘的获取会有一定的延迟，导致判断失误，这边暂时通过增加键盘获取的次数和加上F0xx的方式来扩大判断区域，但是实际还是有一些手速慢了就踩不到。考虑游戏就是以手速为其中一种难度，这个问题不太大。

在原版white Tile小游戏中，最下面一排的黑白格子会根据正确与否变成红色或者绿色，实现起来难度也不大，就是在判断的时候增加一个判断reg的输出，进行颜色的转换，但是由于时间原因没有去实现，有一点遗憾，以及Buzzer没有了就没有正确时的音效感觉也少些什么。

总体设计体验良好，实现效果良好。