课程综合实验——苹果大作战

实验报告

姓名：葛帅琦 学号：3150102193 专业：计算机科学与技术

课程名称：计算机组成 同组学生姓名：

实验时间：2017-06-16 实验地点：紫金港东 4-509 指导老师：施青松

一、实验目的和要求

1. 设计简单的总线接口
2. 支持接口功能
   - 主存储器
   - 七段显示器
   - GPIO 独立按键
   - 简单计数器
3.

基本要求：构建 SOC 应用系统
设计一个简单有意义的程序

高级要求：拓展阵列键盘总线接口
拓展 VGA 总线接口（支持 640*480 标准模式）并设计驱动
拓展 PS2 键盘接口并设计驱动

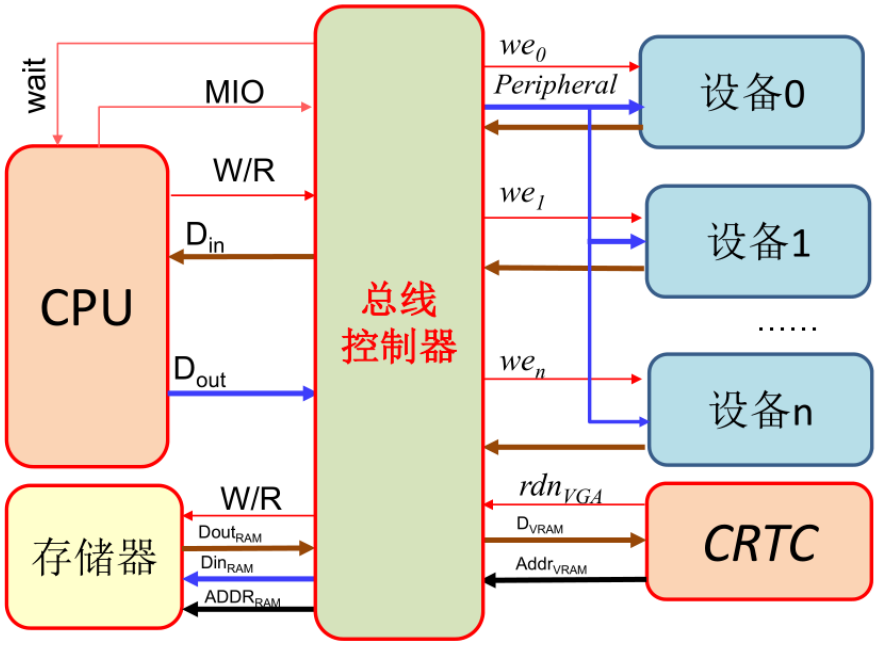
二、实验任务和原理

本小节详细说明实验任务和实验原理，必要时应有图片、表格等。如果内容比较多，可以分节描述，小节的格式如下：

2.1 设计 VGA 显示以及 PS2 控制的游戏

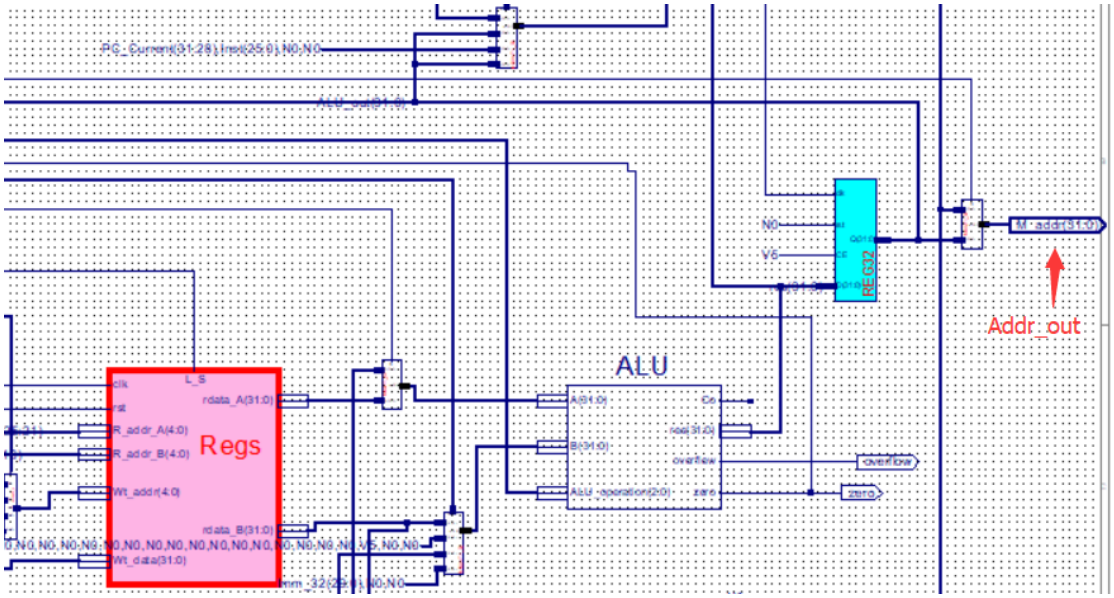
根据系统组要求，利用现有 SOC 平台接入 PS2 以及 VGA，设计一个简单的游戏。

2.2 总线控制原理



总线控制示意图

BUS 总线关键在于 CPU 的 Addr_out 出口，参见下图数据通路



例如 sw 和 lw 指令

sw $s1,0($t1)

lw $s2,0($t2)

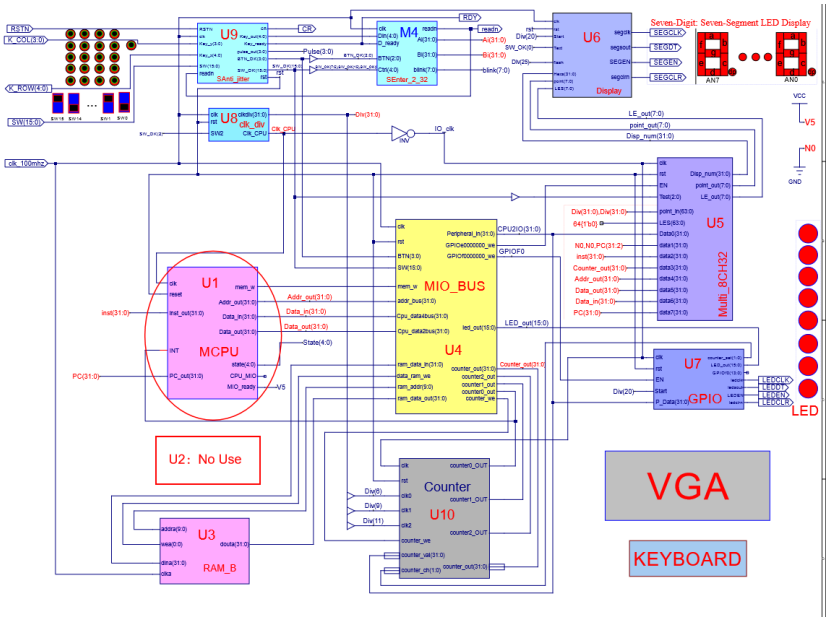
这两天指令首先计算出地址，传输到 bus 当中，bus 接受地址，根据地址调整通往 cpu 或者外设设备的接口，好比是《哈利波特》霍格沃兹学校礼堂中会动的电梯。在 bus 中分配 ps2，vga 以及其他外设设备的接口，从而完成 cpu 与其他设备的交互。

在我的 bus 中，地址如下

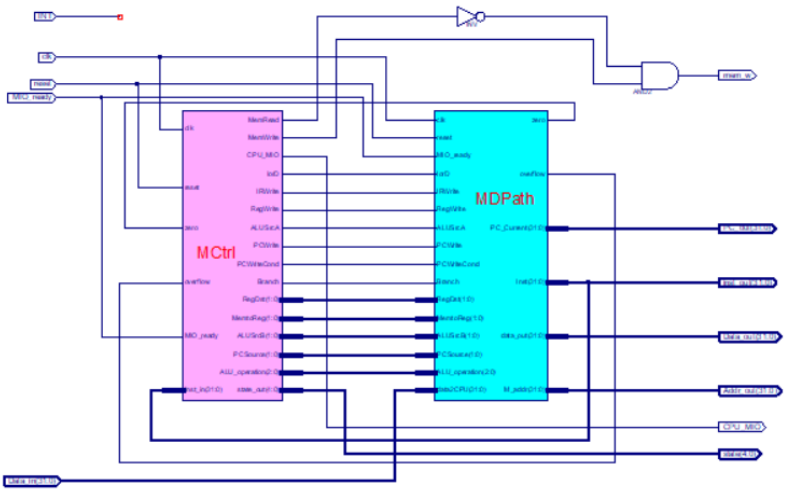
4’h00000000～4’h000004ff 主程序
4’h00000500 applex
4’h00000600 PS2
4’h00000700 appley
4’h00000800 backetx
4’h00000900 分数计数器

2.3 核心硬件架构

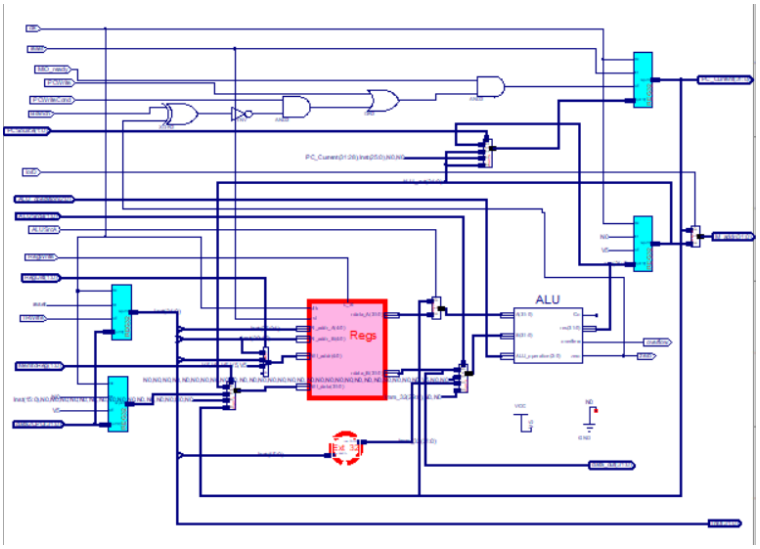
TOP 顶层架构



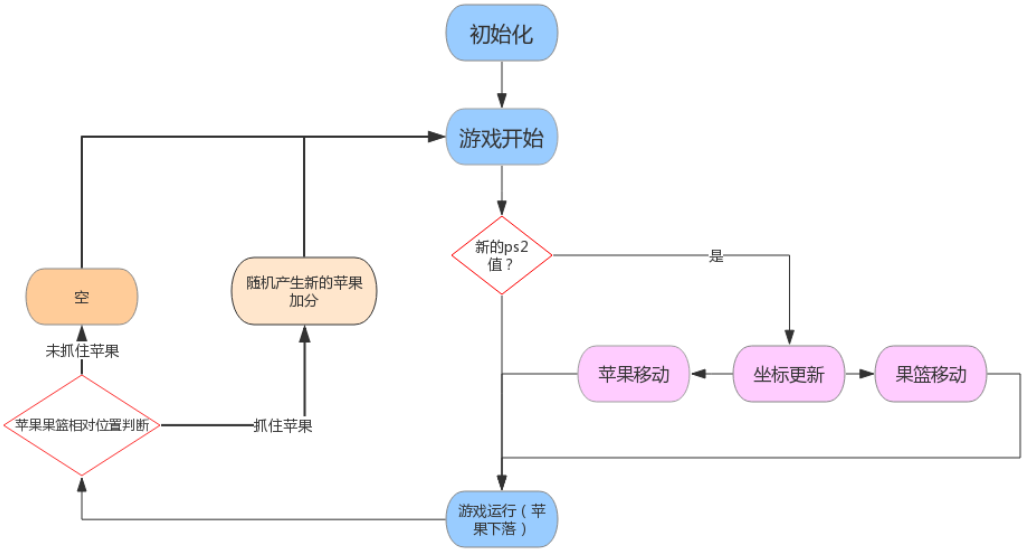
CPU 结构



数据通路



2.4 游戏逻辑状态控制



游戏状态机流程图

三、主要仪器设备

必须采用编号样式，设备的数量和单位应对齐。示范如下：

1. Swrod 开发板　　1 套
2. win10 PC 机　　1 台

3. ISE14.7 …

四、实验实现方法、步骤与调试

BUS 源码

```
module MIO_BUS(
        input clk,
        input rst,
        input [3:0] BTN,    // 4
        input [7:0] SW,     // 8
        input mem_w, //   CPU
        input [31:0] Cpu_data2bus,   //data from CPU
        input [31:0] addr_bus,       //addr from 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,   // 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,               // RAM   RAM
        output reg GPIOf0000000_we,         // GPIOffffff00_we
        output reg GPIOe0000000_we,         // GPIOfffffe00_we
        output reg counter_we,               //记数器
        output reg [31:0] Peripheral_in,    //送外部设备总线

        input [31:0] lg_out,
        output reg lg_we,
        output reg [6:0] lg_addr,



        input wire [15:0] xkey, // ps2

        output reg [8:0]bus4game,
        output reg axwe,
```

```
        output reg aywe,
        input wire [8:0] ax,
        input wire [8:0] ay,

        input wire [8:0] bx,
        output reg bxwe,
        output reg [31:0] score_reg
    );

    reg data_ram_rd;
    reg GPIOf0000000_rd;
    reg GPIOe0000000_rd;
    reg counter_rd;
    reg [7:0] led_in;

    reg lg_rd;

    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 写信号
        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;         // 内存物理地址：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:

            if( addr_bus == 32'h00000600 )                  // ps2 address
                begin
                    axwe = 0;
                    aywe = 0;
```

```
                    Cpu_data4bus = {16'h0000,xkey};    // send ps2 to cpu
                end
            else if( addr_bus == 32'h00000500 )          // x address
                begin
                    axwe = mem_w;
                    aywe = 0;
                    Cpu_data4bus = ax ;
                    bus4game = Cpu_data2bus[8:0];   //      busforgame      is      position
cpudata4bus : write to cpu
                end
            else if( addr_bus == 32'h00000700 )          // y address
                begin
                    aywe = mem_w;
                    axwe = 0;
                    Cpu_data4bus = {23'h000000,ay};
                    bus4game = Cpu_data2bus[8:0];
                end

            else if ( addr_bus == 32'h00000800 )          // x address of backet
                begin
                    bxwe = mem_w;
                    Cpu_data4bus = {23'h000000,bx};
                    bus4game = Cpu_data2bus[8:0];
                end

            else if ( addr_bus == 32'h00000900 )
                begin
                    if(mem_w)
                        score_reg = Cpu_data2bus;
                    else
                        Cpu_data4bus = score_reg;
                end

            else
                begin
                axwe = 0;
                aywe = 0;
                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
            4'hd: begin      // Life game
```

```
                    lg_we = mem_w;
                    lg_addr = addr_bus[6:0];
                    Peripheral_in = Cpu_data2bus;
                    Cpu_data4bus = lg_out;
                    lg_rd = ~mem_w;
                end
                4'he: begin                         // 七段显示器 (e0000000 - effffff, SSeg7_Dev)
                    GPIOe0000000_we = mem_w;
                    Peripheral_in = Cpu_data2bus;
                    Cpu_data4bus = counter_out;
                    GPIOe0000000_rd = ~mem_w;
                end
                4'hf: begin                         // PIO (f0000000 - fffffff0, 8 LEDs & counter, f000004-fffffff4)
                    if (addr_bus[2]) begin                          //f0000004
                        counter_we = mem_w;
                        Peripheral_in = Cpu_data2bus;
                        Cpu_data4bus = counter_out;     //write Counter Value
                        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

            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
```

程序汇编

```
j Initial
j Initial
j Initial
add $t5, $zero, $zero
add $t5, $zero, $zero
add $t5, $zero, $zero
add $t5, $zero, $zero
add $t5, $zero, $zero
add $t5, $zero, $zero
add $t5, $zero, $zero
Initial:
add $t5, $zero, $zero
add $t3, $zero, $zero    // t3 is clock count
addi $s0, $zero, 1280    // s0 is x address h500
addi $s1, $zero, 1792    // s1 is y address h700
addi $s2, $zero, 2048    // s2 is x of backet h800
addi $s7, $zero, 1536    // s7 is ps2 address h600
addi $t8,$zero,15        // t8 is score reg
addi $t9,$zero,2304            // t9 is addr of score reg
addi $s4,$zero,0               // s4 is random num
sw $t8,0($t9) // store score
start:                         // set seed of random_
add $t6,$t6,$zero
addi $s4,$s4,7
addi $t6,$s4,-490
bne $t6,$zero,next
addi $s4,$zero,0
next:
lw $t7, 0($s7)                  // t7 = ps2 value
lw $t0, 0($s0)                  // t0    = x value
lw $t1, 0($s1)                  // t1 = y value
lw $t2, 0($s2)            // t2 = x backet
addi $t6,$t3, -10                                          // clock
bne $t6,$zero, appledown
addi $t6, $t7, -7196            // t6 = ps2 - 1c1c (A)
beq $t6, $zero, Left      // if t6 == 0 go left
addi $t6, $t7, -8995    // t6 = ps2 - 2323 (D)
beq $t6, $zero, Right // if t6 == 0 go right
addi $t6, $t7, -7453    // t6 = ps2 - 1d1d (W)
beq $t6, $zero, Up // if t6 == 0 go up
addi $t6, $t7, -6939    // t6 = ps2 - 1b1b (S)
beq $t6, $zero, Down // if t6 == 0 go right
addi $t6, $t7, -8738 // t6 = ps2 - 1a1a(X)
beq $t6, $zero, BLeft // if t6 == 0 go right
```

```
addi $t6, $t7, -8481 // t6 = ps2 - 2121(C)
beq $t6, $zero, BRight // if t6 == 0 go right
appledown:
add $t6,$t6,$zero
addi $t3, $t3, 1                                        // count clock
addi $t6, $t3, -30   // every 30 cycle, y + 3
beq $t6, $zero, changey// when t3 = 30
j start
BLeft:
add $t6,$t6,$zero
lw $t6,0($s2)
addi $t6,$t6, -1
sw $t6,0($s2)
j start
BRight:
add $t6,$t6,$zero
lw $t6,0($s2)
addi $t6,$t6, +1
sw $t6,0($s2)
j start
changey:
add $t6,$t6,$zero
add $t3, $zero, $zero    // clear clock
addi $t1, $t1, 1         // y+1
addi $t5,$t1,-400        // if t1 == 400
bne $t5,$zero,swy        // y != 400, j swy
slt     $t5,$t2,$t0      // t2(bx) > ax t0
beq $t5,$zero,swy
addi $t5,$t2,50
slt      $t5,$t0,$t5     // bx + 150 > ax + 100
beq $t5,$zero,swy
add $t6,$t6,$zero
lw $t8,0($t9)
addi $t8,$t8,4           // add score
add $t1,$zero,$zero          // clear y
random:
add $t0 ,$t0, $s4                // set random x
addi $t6,$zero,540
slt   $t6,$t6,$t0
bne $t6,$zero,randomnext
addi $t0,$t0,-300
randomnext:
sw $t1, 0($s1) // sw y
sw $t0, 0 ($s0)    // sw x
```

```
add $t6,$t6,$zero
sw $t8,0($t9)                // store score
swy:
add $t6,$t6,$zero
sw $t1, 0($s1)      // sw y
j start
Left:
add $t6,$t6,$zero
addi $t0, $t0, -5 // x - 5
sw $t0, 0 ($s0)    // sw x
j start
Right:
add $t6,$t6,$zero
addi $t0, $t0, 5 // x + 5
sw $t0, 0($s0)    // sw x
j start
Up:
add $t6,$t6,$zero
addi $t1, $t1, 5 // y + 5
sw $t1, 0($s1) // sw y
j start
Down:
add $t6,$t6,$zero
addi $t1, $t1, -5 // y-5
sw $t1, 0($s1) // y+5
END:
j start
```

ps2 代码

```
module ps2_receiver(
    input wire clk, clr,
    input wire ps2c, ps2d,
    output wire [15:0] xkey,
    output reg [15:0] regkey
    );

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

    assign xkey = {shift2[8:1], shift1[8:1]};
```

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

always @ (posedge clk or posedge clr)
begin
	if (clr == 1)
	begin
		ps2c_filter <= 0;
		ps2d_filter <= 0;
		PS2Cf		<= 1;
		PS2Df		<= 1;
	end
	else
	begin
		ps2c_filter <= {ps2c, ps2c_filter[7:1]};
		ps2d_filter <= {ps2d, ps2d_filter[7:1]};
		if (ps2c_filter == 8'b1111_1111)
			PS2Cf <= 1;
		else if (ps2c_filter == 8'b0000_0000)
			PS2Cf <= 0;
		if (ps2d_filter == 8'b1111_1111)
			PS2Df <= 1;
		else if (ps2d_filter == 8'b0000_0000)
			PS2Df <= 0;
	end
end

// Shift register used to clock in scan codes from PS2
always @ (negedge PS2Cf or posedge clr)
begin
	if (clr == 1)
	begin
		shift1 <= 0;
		shift2 <= 1;
	end
	else
	begin
		shift1 <= {PS2Df, shift1[10:1]};
		shift2 <= {shift1[0], shift2[10:1]};
	end
end
```

```
endmodule
```

VGA 代码

```
module vga_controller(
    input wire clk, clr,
    output reg hsync, vsync,
    output wire video_on,
    output wire [9:0] pixel_x, pixel_y
    );

    parameter hpixels = 800;
    parameter vlines    = 525;
    parameter hbp         = 143;
    parameter hfp         = 783;
    parameter vbp         = 31;
    parameter vfp         = 519;

    reg [9:0] hc, vc;

    assign pixel_x = hc - hbp - 1;
    assign pixel_y = vc - vbp - 1;


    always @ (posedge clk or posedge clr)
    begin
        if (clr == 1)
            hc <= 0;
        else
        begin
            if (hc == hpixels - 1)
            begin
                hc <= 0;
            end
            else
            begin
                hc <= hc + 1;
            end
        end
    end

    always @*
```

```
    begin
        if(hc >= 96)
            hsync = 1;
        else
            hsync = 0;
    end

    always @(posedge clk or posedge clr)
    begin
        if (clr == 1)
        begin
            vc <= 0;
        end
        else
        begin
            if (hc == hpixels - 1)
            begin
                if (vc == vlines - 1)
                begin
                    vc <= 0;
                end
                else
                begin
                    vc <= vc + 1;
                end
            end
        end
    end

    always @*
    begin
        if(vc >= 2)
            vsync = 1;
        else
            vsync = 0;
    end

    assign video_on = (hc < hfp) && (hc > hbp) && (vc < vfp) && (vc > vbp);

endmodule
```

本实验图片均放弃数逻读取 bmp 文件 rgb 三通道模式，而是使用 python 读取 png 的 rgba 四通道模式，从而灵巧的回避素材图片的背景问题，从而显示更加精美的图片。Python 源码如

下

Python 读图：

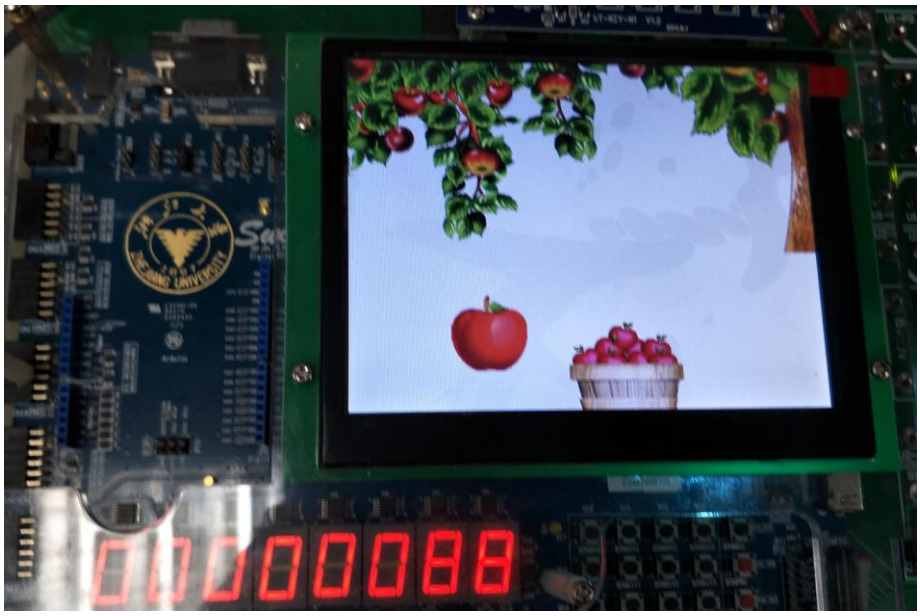
```
import numpy as np
from scipy import misc
import os, sys

for infile in sys.argv[1:]:
    # read the image from the file
    img = misc.imread(infile)
    (row, col, tunnel) = img.shape
    print img
    # matrix transpose
    red = np.transpose(img[:, :, 0])
    green = np.transpose(img[:, :, 1])
    blue = np.transpose(img[:, :, 2])

    if img.shape[2]>3:
        alpha = np.transpose(img[:, :, 3])
    else:
        alpha = np.zeros(red.shape)
    # create .COE file
    f, e = os.path.splitext(infile)
    outfile = f + '.coe'
    print img.shape
    try:
        file = open(outfile, 'w')
        file.write('memory_initialization_radix=16;\n')
        file.write('memory_initialization_vector=\n')
        # write image data to file
        print row, col
        for i in range(row - 1):
            for j in range(col):
                data = ''.join('%x%x%x%x,' % (red[j][i] / 16, green[j][i] / 16, blue[j][i] / 16,
alpha[j][i] / 16))
                file.write(data + '\n')
        for j in range(col - 1):
            data = ''.join('%x%x%x%x,' % ( \
                red[j][row - 1] / 16, green[j][row - 1] / 16, blue[j][row - 1] / 16, alpha[j][row
- 1] / 16))
            file.write(data + '\n')
        # last data value supposed to have a semicolon instead of a comma
        data = ''.join('%x%x%x%x;' % ( \
            red[col - 1][row - 1] / 16, green[col - 1][row - 1] / 16, blue[col - 1][row - 1] / 16, \
            alpha[col - 1][row - 1] / 16))
```

```
            file.write(data)
            print "finish"
        except IOError:
            print('cannot open', infile)
        file.close()
```

五、实验结果与分析



VGA 显示游戏界面，两个主要元素是苹果和果篮。
数码管显示当前得分，SW[7:5] = 2’b010, SW[0] = 1

游戏规则是移动果篮接受苹果，收到苹果进行加分。

显示器已加分。

通过控制 SWAD 控制苹果的移动，按 XC 控制果篮的移动。

Note： PS2 的读取采取的是轮询策略，所以按键有些卡。并且一个方向不怎么卡，另一个方向移动比较卡顿。

六、不足与展望：

1. CPU 供支持指令不够丰富，缺少乘法指令（尽管 MIPS 原版没有这条指令）以及例如 sll，move 等较为实用的指令。由于制作者 CPU 中 alu，datapate 最初都是体系绘图连接而非 Verilog 编写，在 alu 八个功能上再拓展对 alu 外层控制改动较为费时，临近期末时间有限无法完成。

2. 对外部设备的控制采取的是指令轮询的策略而非中断机制，导致 CPU 有相当一部分时间在做无效指令的空转，利用率较低。

3. VGA 的实现并未严格的按照现有的计算机体系结构，用 VGA 读取 VRAM 来实现。原因有二：一是缺乏乘法指令，在用汇编实现 640*480 的像素数组更新时，所需的汇编语句极为复杂。二是考虑如果用 CPU 来实现 VRAM 的更新，将需要大于 307200 条指令完成数组的遍历，将严重影响程序性能。目前采取的指令轮询外设的策略已经使 PS2 较为卡顿并且影响苹果自由下落。

4. 以上问题将在暑期继续改进。

七、讨论、心得

选计组前听说楼学庆老师的班级是最累的，上完这学期的计组课，发现施老师的班才是最累的。别的计组班还在做单周期 CPU 中断的时候，我们班已经做完多周期 CPU 的实验，别的班做完多周期实验，所有实验也就结束了；姜晓红老师的班稍微多点任务，做个乘法器的 bonus，但是施老师竟然要做个简单的应用，系统组还要接入 PS2 和 VGA 做个小游戏，简直“丧心病狂”~，在期末众多作业压身的情况下，感觉世界崩塌了一样。

一开始一直怀疑自己能不能、有没有时间来完成这个 project，当时心里很多疑惑，不知道怎么在体系里面接 VGA，PS2，怎么输入输出这些数据到 CPU，怎么写汇编程序，怎么控制 bus，怎么调试，谁来生成 coe 等等。后来自己一步步摸索，较早就完成了这个 project，期间遇到了无数个问题，踩了很多坑又从里面爬出来，还通宵了一次，从晚上八点一直做到第二天早上六点，看着外面天色逐渐亮了起来。

我在 6 月 15 号完成了 project，做出了这个苹果大作战的小游戏，完成时间还是比较早的。做完 project 还是很有成就感的，也有很多的收获，中间遇到的许多问题，本质上是我对 CPU 的理解还不够到位，通过这次实验，对 CPU 的运作更加熟悉了。