# 实验十二（大实验）报告—— 打字小游戏

**191220185 朱嘉琦**

## 一、实验目的

本实验用 FPGA 实现计算机系统基础课 PA4 中的打字小游戏，即屏幕上随机产生一些字母，并以不同随机速度掉落。玩家需要按对应的键在字符掉落到屏幕底部之前消除字符，参考实现如图12-1所示：

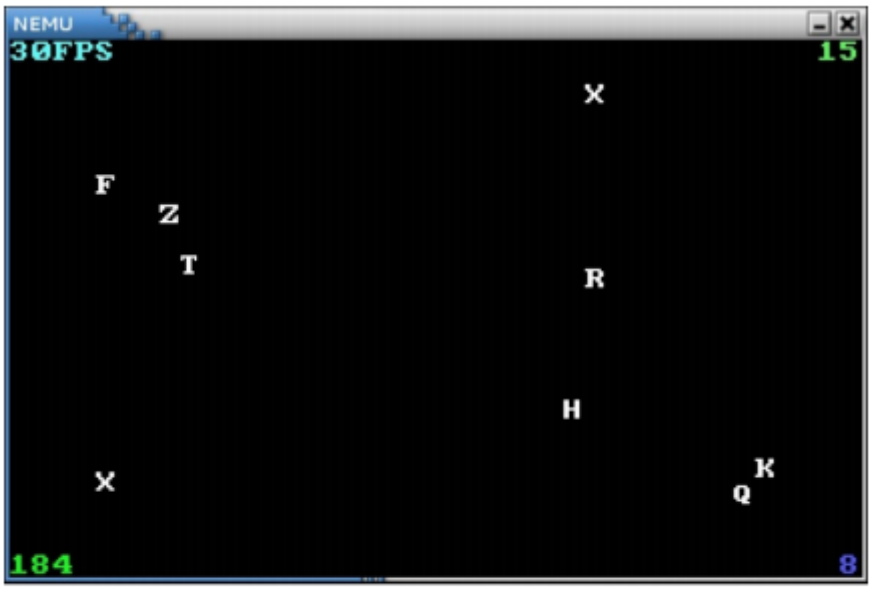



图 12-1: 打字小游戏界面

本实验主要利用前面实现过的键盘的输入识别与响应和显示器的输入与显示来综合设计一个复刻打字小游戏，利用模块划分和顶层设计等方法构架整个工程的代码，使得代码的整体架构更加明了，增加代码的易读性和可扩展性。通过该系统的实现深入理解多个模块之间的交互和接口的设计。

## 二、游戏说明书

**开始游戏：**

导入电脑之后直接开始，可通过sw[1]重启游戏，sw[0]仅关闭显示屏。

**操作说明：**

通过键盘输入屏幕上出现的英文字母，消除掉落的字符。（如有相同的字符靠左字符优先）

**游戏信息提示：**

左下角为得分计数，右下角为失误计数，右上角为计时，左上角为屏幕刷新率：60FPS。

## 三、实验原理（背景知识）

### 随机数发生器

主要是利用线性反馈移位寄存器，在时钟的触发沿，根据其控制信号，将储存在其中的数据向某个方向移动，从而获得不同的状态，参考教科书第 534 页 LFSR 反馈方程（表8-26）。移位寄存器是由n个D触发器组成，而n个触发器可以提供$2^n - 1$个状态，其中不包括全零的状态，再通过触发器和异或门的组合设计反馈系数，使得每次反馈系数不同时状态转移图也不同，这样便产生了一个伪随机数发生器。

**表8-26　线性反馈移位寄存器计数器的反馈方程**

| $n$ | 反馈方程 |
|---|---|
| 2 | $X2 = X1 \oplus X0$ |
| 3 | $X3 = X1 \oplus X0$ |
| 4 | $X4 = X1 \oplus X0$ |
| 5 | $X5 = X2 \oplus X0$ |
| 6 | $X6 = X1 \oplus X0$ |
| 7 | $X7 = X3 \oplus X0$ |
| 8 | $X8 = X4 \oplus X3 \oplus X2 \oplus X0$ |
| 12 | $X12 = X6 \oplus X4 \oplus X1 \oplus X0$ |
| 16 | $X16 = X5 \oplus X4 \oplus X3 \oplus X0$ |
| 20 | $X20 = X3 \oplus X0$ |
| 24 | $X24 = X7 \oplus X2 \oplus X1 \oplus X0$ |
| 28 | $X28 = X3 \oplus X0$ |
| 32 | $X32 = X22 \oplus X2 \oplus X1 \oplus X0$ |

## 键盘的输入和识别

键盘处理器在识别到有键按下或放开后会从PS2_DAT引脚送出扫描码。按键按下时送出的扫描码为”通码“，释放时送出的扫描码为”断码“，每个键都有唯一的通码和断码。多个键按下时会逐个输出扫描码。键盘向主机传送数据时是通过PS2_DAT和PS2_CLK信号的控制，当其都为高电平时键盘才可发送信号。接受键盘数据时使用fifo队列，防止数据丢失，fifo配有写指针和读指针。处理数据时首先在接受数据后放入缓冲区，收集完11位后转移至fifo。当fifo不空时送出ready信号，表示有键按下；当队列溢出时，送出overflow信号。系统在ready的情况下读取键盘数据。读取后将nextdata_n置零，并将读指针前移，读取下一数据。

## 显示屏的输入和显示

### 字符显示——字模

字符显示界面只在屏幕上显示 ASCII 字符，其所需的资源比较少。首先， ASCII 字符用 7bit 表示，共 128 个字符。大部分情况下，我们会用 8bit 来表示单个字符，所以一般系统会预留 256 个字符。我们可以在系统中预先存储这 256 个字符的字模点阵，如图 11-1所示。

这里每个字符高为 16 个点，宽为 9 个点。因此单个字符可以用 16 个 9bit 数来表示，每个 9bit 数代表字符的一行，对应的点为 “1” 时显示白色，为 “0” 时显示黑色。因此，我们只需要 256 × 16 × 9 ≈37kbit 的空间即可存储整个点阵。例如图11-2就是字模"A"与储存器的关系。

图 11-1: ASCII 字符字模

**扫描显示**

我们之前已经实现了 VGA 控制模块，该模块可以输出当前扫描到的行和列的位置信息，我们只需要稍加改动，即可让其输出当前扫描的位置对应 30×70 字符阵列的坐标 (0 ≤ x ≤ 69, 0 ≤ y ≤ 29)。利用该坐标，我们可以查询字符显存，获取对应字符的 ASCII 编码。利用 ASCII 编码，我们可以查询对应的点阵 ROM，再根据扫描线的行和列信息，可以知道当前扫描到的是字符内的哪个点。这时，可以根据该点对应的 bit 是 1 还是 0，选择输出白色还是黑色。（如图11-3所示）

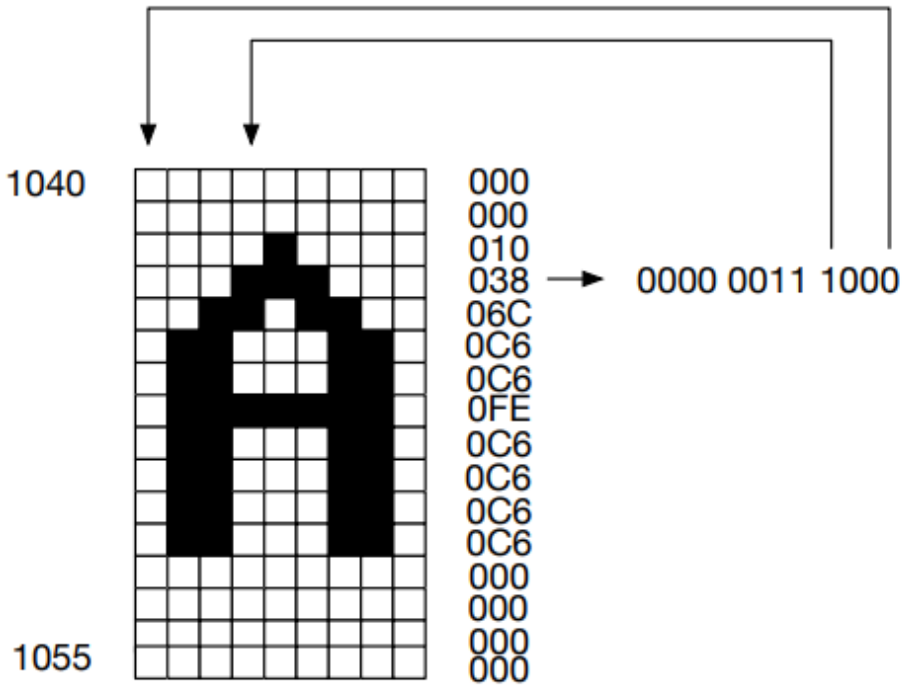



图 11-2: 字模 “A” 与存储器的关系

因此，将显示的**过程**如下：

1. 根据当前扫描位置，获取对应的字符的 x,y 坐标，以及扫描到单个字符点阵内的行列信息。
2. 根据字符的 x,y 坐标，查询字符显存，获取对应 ASCII 编码。
3. 根据 ASCII 编码和字符内的行信息，查询点阵 ROM，获取对应行的 9bit 数据。
4. 根据字符内的列信息，取出对应的 bit，并根据该 bit 设置颜色。此处可以显示黑底白字或其他彩色字符，只需要按自己的需求分别设置背景颜色和字符颜色即可。

# 四、实验环境/器材

## 实验环境：

Quarters 17.1 Lite

## 实验器材

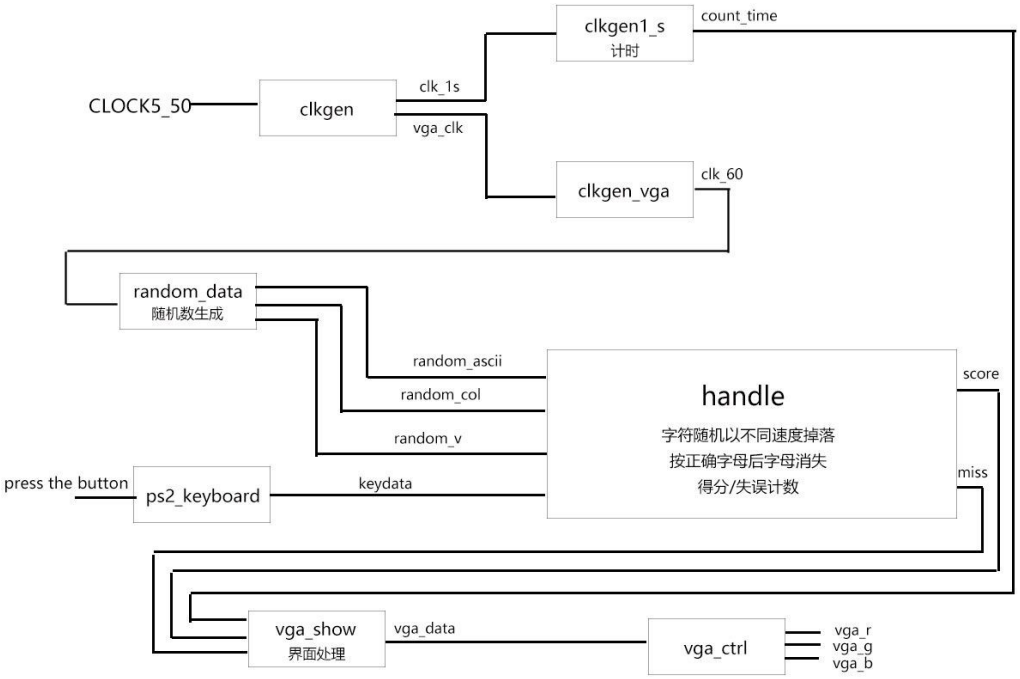
DE10-Standard 开发板、FPGA芯片，VGA显示器，PS/2键盘

# 五、实验设计思路

## 1. 设计目标

- 用 FPGA 实现计算机系统基础课 PA4 中的打字小游戏，即屏幕上随机产生一些字母，并以不同随机速度掉落。玩家需要按对应的键在字符掉落到屏幕底部之前消除字符。
- VGA采用60Hz刷新率，分辨率和字符大小同之前的实验一致。（分辨率为$640 * 480$，每个字模的大小为$9 * 16$像素点）
- 实现随机字符产生，字符掉落，实现字符的平滑下落，按键消除字符功能和游戏逻辑，同时每个字符有随机的下落速度。
- 游戏信息提示，如 “FPS” 等字符，色彩可以自选。

## 2. 整个工程的构建及模块间的大致关系



整个工程为final模块，在该模块内对其他模块进行了实例化，模块主要分为：分频模块、handle模块、显示模块和随机数生成模块：

1. **分频模块：**通过系统时钟，产生一个1s分频器和vga驱动时钟和信号，以及一个随机数模块的seed。（与时钟有关）
2. **随机数生成模块：**根据输入的original源数据生成一个与时钟有关的数，由于时钟频率很高，所以可以作为一个伪随机数生成模块。
3. **Handle模块：**包含游戏的初始化，对字符进行处理。根据随机数生成模块提供的随机列和随机速度使随机字母在随机位置以不同速度掉落、按键正确后离屏幕最下方最近的一个对应字符消失并进行得分计数，如果有字母掉出屏幕外则进行miss计数。
4. **显示模块：**字符掉落显示，四个角分别显示分数、时间、“60FPS”、miss数量。

## 3. 重难点及代码实现分析

### 寻址方式

在之前字符输入界面的中，将$640*480$的分辨率分为30行70列，一开始使用这样的方法显示，发现这样下落非常的卡顿，因为字符在每一列上只会出现在固定的30个位置上，每次下落即16个像素点，无论速度再快都有点别扭。因此，修改了寻址方式，将屏幕分成70列，以一个个像素点分行，也就是480行。用71个10位寄存器存储该列的ascii码，用71个10位寄存器存储该列的行地址。用71个4位寄存器存储该列具有的速度，用71个1位寄存器存储该列的显示位（故意开大了一点点）。也就是每次vga扫描，通过vga_haddr得到列地址，看显示位是否为1，如果为1就处理，通过vga_vaddr-该列存储的行地址和对应的ascii码来显示。对应的字符下落就是下落一个像素点，即行地址+1。这个方式唯一有个弊端即该列只能出现一个字符，不能同时出现。

```
//////////寻址结构//////////
wire [9:0] mem_ascii [71:0];//某列的ascii码
wire [9:0] mem_row[71:0];//某列的行地址
wire [3:0] mem_v[71:0];//速度
wire [3:0] mem_alv[71:0];//积累量 满v则下移一个像素点
wire mem_valid[71:0];//是否显示位
```

### 字符显示

字符显示基本与之前的VGA控制实验一致，核心显示代码一致，如下：

```
if(c >= 5 && c < 68)
    begin//字符掉落界面
        if(flag_word && mem_valid[c])
        begin
            ascii_temp = mem_ascii[c];
            rom_temp = rom[{ascii_temp,x}];
            vga_data =(rom_temp[y]==1)?24'h000000:24'hFFFFFF;
        end
        else
            vga_data = 24'h000000;
    end
```

### 字符的平滑下落

mem_v储存随机生成的字符下落速度（采用不同时间对应不同速度），mem_alv累计单位时间，每到clk_60的下降沿时增加一，直到到达mem_v中的时间时mem_row下降一个像素点（其中mem_row储存行地址，mem_v储存速度，mem_alv储存积累量）

```
if (mem_row[j]<479) //字符未下落最下面
    begin
        if (mem_alv[j]==mem_v[j])//达到速度，向下移动1个像素点
        begin
            mem_row[j]=mem_row[j]+1;
            mem_alv[j]=0;
        end
        else begin//未达到速度，alv+1，继续积累
            mem_alv[j]=mem_alv[j]+1;
        end
    end
```

#### 字符的消除

当字符在落出屏幕外该字母在键盘上被按下，则需要将其消除。根据ps/2键码输出的特性，flag_handle为之前对键码的判断。mem_ascii储存某列的ascii码，通过判断是否与按键的ascii码相同并且按键已经松开进行消除。

```
if (mem_ascii[j]==key2ascii[keydata[7:0]]&&flag_handle==0)//消除字符
    begin
        mem_valid[i]=0;
        mem_ascii[i]=0;
        mem_v[i]=0;
        mem_row[i]=0;
        mem_alv[i]=0;
        score=score+1;//分数加1
        flag_handle=1;
    end
```

#### 随机列产生随机ascii码和随机速度

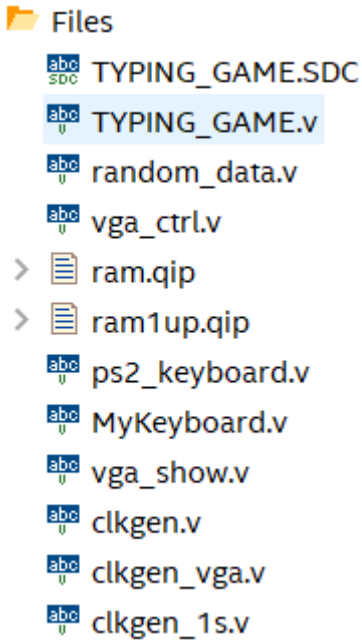
因为四个角放了东西，所以字符掉落范围在5-68行，如果该列显示位为0，即在该列生成随机字符，这样避免一列出现两个（否则之后消除时不好操作）。

```
if (count4==200)//在随机列生成随机速度下落的字符
    begin
        if (mem_valid[5+random_col%63]==0)
            begin
                mem_valid[5+random_col%63]=1;
                mem_ascii[5+random_col%63]=8'h61+random_ascii%26;
                mem_row[5+random_col%63]=0;
                mem_v[5+random_col%63]=random_v%8;
                mem_alv[5+random_col%63]=0;
            end
    end
```

# 五、核心代码

## 1. 工程文件

本次实验涉及到多个模块




顶层文件核心代码

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module TYPING_GAME(

    ////////////// CLOCK //////////
    input                       CLOCK2_50,
    input                       CLOCK3_50,
    input                       CLOCK4_50,
    input                       CLOCK_50,

    ////////////// SW //////////
    input             [9:0]     SW,

    ////////////// VGA //////////
    output                      VGA_BLANK_N,
    output            [7:0]     VGA_B,
    output                      VGA_CLK,
    output            [7:0]     VGA_G,
    output                      VGA_HS,
    output            [7:0]     VGA_R,
    output                      VGA_SYNC_N,
    output                      VGA_VS,

    ////////////// PS2 //////////
    inout                       PS2_CLK,
    inout                       PS2_CLK2,
    inout                       PS2_DAT,
    inout                       PS2_DAT2,
);



//=======================================================
```

```
//  REG/WIRE declarations
//=============================================================

assign VGA_SYNC_N=0;
wire clkreset;
assign clkreset = SW[0];
wire reset;
assign reset = SW[1];

//////////寻址结构//////////
wire [9:0] mem_ascii [71:0];//某列的ascii码
wire [9:0] mem_row[71:0];//某列的行地址
wire [3:0] mem_v[71:0];//速度
wire [3:0] mem_alv[71:0];//积累量 满v则下移一个像素点
wire mem_valid[71:0];//是否显示位



//=============================================================
//  Structural coding
//=============================================================
//////////键码转换为ascii码/////////
reg [7:0] key2ascii [255:0];
initial begin
    $readmemh("D:/My_design/final/scancode.txt", key2ascii, 0, 255);
end

//////////产生频率//////////

wire clk_1s;
wire clk_60;
clkgen #(25200000) vgaclk(.clkin(CLOCK5_50),.rst(clkreset),.clken(1'b1),
.clkout(vga_clk));

clkgen_vga #(3000) clk_x1(.clkin(vga_clk),.rst(clkreset),.clken(1'b1),
.clkout(clk_60));

clkgen #(30) time_clk(.clkin(CLOCK5_50), .rst(clkreset), .clken(1'b1),
.clkout(clk_1s));

//////随机数生成////////

wire [7:0] random_ascii;
wire [7:0] random_col;
wire [7:0] random_v;

random_data ascii(clk_60,reset,8'h61,random_ascii);
random_data col(clk_60,reset,8'h30,random_col);
random_data v1(clk_60,reset,8'h35,random_v);


//////////计时//////////
wire [9:0] count_time;
clkgen_1s x2(clk_1s,reset,count_time);

/////////////////vga显示/////////////////
wire [9:0] vga_haddr;
wire [9:0] vga_vaddr;
```

```
wire [23:0] vga_data;
wire [9:0]score;
wire [9:0]miss;

vga_ctrl my_vga(.pclk(VGA_CLK), .reset(reset), .vga_data(VGA_DATA),
.h_addr(vga_haddr),
.v_addr(vga_vaddr), .hsync(VGA_HS), .vsync(VGA_VS), .valid(valid),
.vga_r(VGA_R),
.vga_g(VGA_G), .vga_b(VGA_B));
vga_show
vga_handle(VGA_CLK,score,miss,vga_haddr,vga_vaddr,mem_ascii,mem_row,mem_valid,VGA_DATA);

//////////handle///////////////
/*
 *负责所有字符的处理，代码比较冗长，核心功能代码上面已经给出，功能包含：
 *字符的初始化，字符的下落，字符的消除，miss计数，分数计数，
 *在随机列生成随机速度下落的字符等等
*/
endmodule
```

## 2. VGA控制模块

### 1）vga_ctrl.v

VGA控制模块与显示图片实验保持一致，所以在此仅给出接口声明

```
module vga_ctrl(
    input pclk, //25MHz时钟
    input reset, //置位
    input [11:0] vga_data, //上层模块提供的VGA颜色数据
    output [9:0] h_addr, //提供给上层模块的当前扫描像素点坐标
    output [9:0] v_addr,
    output hsync, //行同步和列同步信号
    output vsync,
    output valid, //消隐
    output [3:0] vga_r, //RGB
    output [3:0] vga_g,
    output [3:0] vga_b
    );
endmodule
```

### 2) clkgen.v

```
module clkgen(
    input clkin,
    input rst,
    input clken,
    output reg clkout
    );
    parameter clk_freq=1000;
    parameter countlimit=50000000/2/clk_freq; // count
```

```
    reg[31:0] clkcount;
    always @ (posedge clkin)
        if(rst)
            begin
                clkcount=0;
                clkout=1'b0;
            end
        else
            begin
                if(clken)
                    begin
                        clkcount=clkcount+1;
                        if(clkcount>=countlimit)
                            begin
                                clkcount=32'd0;
                                clkout=~clkout;
                            end
                        else
                            clkout=clkout;
                    end
                else
                    begin
                        clkcount=clkcount;
                        clkout=clkout;
                    end
            end
endmodule
```

## 界面显示部分代码

将当前的游戏时间，屏幕刷新率，miss数量，得分显示在四个角落，并且根据当前存储的字符的位置信息根据字模点阵显示在显示器上：

```
/***********字模点阵**********/
reg [11:0] rom [4097:0];
initial begin
    $readmemh("D:/My_design/final/vga_font.txt", rom, 0, 4097);
end

wire[11:0]c;
wire[3:0]x;
wire[3:0]y;
reg[7:0]ascii_temp;
reg [11:0]rom_temp;
wire flag_word;
assign c=vga_haddr/9;
assign x=vga_vaddr-mem_row[c];
assign y=vga_haddr%9;
assign flag_word=(((vga_vaddr-mem_row[c])>=0)&&((vga_vaddr-mem_row[c])<16))?
1:0;//超出就不显示


always @(negedge vga_clk)
begin
    if(c >= 5 && c < 68) begin//字符掉落界面
        if(flag_word && mem_valid[c]) begin
            ascii_temp = mem_ascii[c];
```

```
            rom_temp = rom[{ascii_temp,x}];
            vga_data =(rom_temp[y]==1)?24'h000000:24'hFFFFFF;
        end
        else
            vga_data = 24'h000000;
    end
    else if(c >= 0 && c < 5) begin//左五列
            if(vga_vaddr >= 0 && vga_vaddr <16) begin//左上角显示60FPS
                case(c)
                    0: ascii_temp = 8'h36;
                    1: ascii_temp = 8'h30;
                    2: ascii_temp = 8'h46;
                    3: ascii_temp = 8'h50;
                    4: ascii_temp = 8'h53;
                    default:;
                endcase
                rom_temp = rom[{ascii_temp,vga_vaddr}];
                vga_data =(rom_temp[y]==1)?24'h00FFFF:24'h000000;
            end

            else if(vga_vaddr >= 464 && vga_vaddr <= 479) begin//左下角显示分数
                case(c)
                    0: ascii_temp = 8'h30 + (score/100) % 10;
                    1: ascii_temp = 8'h30 + (score/10) % 10;
                    2: ascii_temp = 8'h30 + score%10;
                    3: ascii_temp = 8'h0;
                    4: ascii_temp = 8'h0;
                    default:;
                endcase
                rom_temp = rom[{ascii_temp,vga_vaddr-464}];
                vga_data =(rom_temp[y]==1)?24'h00FF00:24'h000000;
            end
    end
    else if (c >= 68 && c < 71)begin
        if(vga_vaddr >= 0 && vga_vaddr <16) begin//右上角显示时间
                case(c)
                    68: ascii_temp = 8'h30 + (count_time / 100) % 10;
                    69: ascii_temp = 8'h30 + (count_time / 10) % 10;
                    70: ascii_temp = 8'h30 + count_time % 10;
                    default:;
                endcase
                rom_temp = rom[{ascii_temp,vga_vaddr}];
                vga_data =(rom_temp[y]==1)?24'hFFFF00:24'h000000;
            end

            else if(vga_vaddr >= 464 && vga_vaddr <= 479) begin//右下角显示miss数
                case(c)
                    68: ascii_temp = 8'h30 + (miss / 100) % 10;
                    69: ascii_temp = 8'h30 + (miss / 10) % 10;
                    70: ascii_temp = 8'h30 + miss % 10;
                    default:;
                endcase
                rom_temp = rom[{ascii_temp,vga_vaddr-464}];
                vga_data =(rom_temp[y]==1)?24'hFF0000:24'h000000;
            end
        end
end
```

## 3. 键盘模块

### ps2_keyboard.v

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data);
input clk,clrn,ps2_clk,ps2_data;
output reg[15:0] data;
reg [9:0] buffer;
reg [3:0] count;
reg [2:0] ps2_clk_sync;

always @(posedge clk) begin
        ps2_clk_sync <= {ps2_clk_sync[1:0],ps2_clk};
end

wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];


always @(posedge clk) begin
        if (clrn == 1) begin // reset
                count <= 0;
        end

        else if (sampling) begin
                if (count == 4'd10) begin
                        if ((buffer[0] == 0) && // start bit
                                    (ps2_data) && // stop bit
                                (^buffer[9:1])) begin // odd parity
                                if(data[7:0] == 8'hf0)
                                        data <= {data[7:0],8'hff};
                                else begin
                                        data <= {data[7:0],buffer[8:1]};
                                end
                        end
                        count <= 0; // for next
                end

                else begin
                        buffer[count] <= ps2_data; // store ps2_data
                        count <= count + 3'b1;
                end
        end
end

endmodule
```

## 4. 随机数模块

### random_data.v

根据八位输入的ori源数据生成一个与时钟有关的数，由于时钟频率很高，所以可以作为一个伪随机数生成模块：

```
module random_data(
          input clk_60,
          input reset,
          input [7:0]ori,
          output reg [7:0] random_data
          );

reg [5:0]count;
reg [7:0] in;
always @(negedge clk_60)
begin
    if (reset)
    begin
     random_data<=ori;
     in<=random_data[4]^random_data[3]^random_data[2]^random_data[0];
     count=0;
    end
    else
    begin
    if (count==20)
    begin
        random_data<={in,random_data[7:1]};
        in<=random_data[4]^random_data[3]^random_data[2]^random_data[0]^(~
(random_data[1]|random_data[2]|random_data[3]|random_data[4]|random_data[5]|rand
om_data[6]|random_data[7]));
        count=0;
    end
    else
    count=count+1;
    end
end
endmodule
```

## 五、实验步骤

- 首先去试玩打字小游戏，了解游戏的基本逻辑，将整个工程中几个重要模块抽象出来，主要为VGA显示控制模块，PS/2键盘控制输入模块，字符与屏幕显示模块，随机数生成模块，handle逻辑处理模块，进行模块划分后分工完成不同模块的代码。
- 利用system builder同时接入VGA和PS/2接口，创建工程文件，实例化之前实验做的vga_ctrl控制模块和ps2_keyboard键盘模块，random_data随机数生成模块，利用这些模块在顶层文件的handle部分中实现字符的初始化，字符的下落，字符的消除，miss计数，分数计数，在随机列生成随机速度下落的字符等等功能。
- 由于本实验中需要用到多个不同时序，所以将clk进行分频，分为clk_1s（用于游戏时间的记录），clk_60（用于实现60Hz刷新率的屏幕），vga_clk（VGA的时钟）

```
clkgen #(25200000) vgaclk(.clkin(CLOCK5_50),.rst(clkreset),.clken(1'b1),
.clkout(vga_clk));

clkgen_vga #(3000) clk_x1(.clkin(vga_clk),.rst(clkreset),.clken(1'b1),
.clkout(clk_60));

clkgen #(30) time_clk(.clkin(CLOCK5_50), .rst(clkreset), .clken(1'b1),
.clkout(clk_1s));
```

- 检查是否有语法错误，最终编译通过，连接到FPGA实验板上，屏幕上已经可以正常显示游戏信息以及刷新随机字符，通过键盘输入发现已经可以消除字符，但字符下落卡顿问题仍然存在。
- 更换字符存储的结构后，卡顿问题解决，可以正常游戏。
- 检查是否有语法错误，最终编译通过，连接到FPGA实验板上，硬件验证。

## 六、测试方法

编译生成二进制文件后，在FPGA开发板运行，将键盘和显示屏与FPGA连接，所有功能均符合预期设计（主要在字符是否平滑落下、计数是否正确、消除是否符合要求、是否随机生成），则该打字小游戏正确实现。

## 七、实验结果——打字小游戏

（已经给助教验收）

## 八、思考

**代码的实例化与模块化：**本次实验中主要的实例化还是来自于之前键盘实验和显示器实验保留下来经过实验验证的模块的直接复用，而其实我们自己完成的模块化只有随机数以及显示模块，大部分的逻辑操作（即handle部分）还是在顶层文件中编写的，感觉还是对实例化的理解太少了，经常会出现编译成功但是到板子上之后出现问题的情况，所以为了方便就将大量代码写在了顶层文件中。这样写的代码不方便以后的复用，所以还是需要多加练习，更加深刻的理解代码的实例化于模块化。

## 九、实验中遇到的问题及解决方法

- **编译时间过长：**为了方便指针操作，程序中写了几个二维数组作为rom而不是利用ip核生成，虽然代码层面方便了不少，但是占用资源过多，编译过程非常的慢，特别是当现在项目比较大，需要多次编译烧录检查，这样的编译时间成本就显得有些得不偿失，在以后还是多利用ip核，而不要在代码中声明容量较大的rom。
- **字符的平滑落下问题：**一开始实现的时候想通过速度问题解决一列一列落下时给人的“不平滑感”问题，然后实现过程中发现需要足够快的速度才能消除，但导致游戏很难实际操作。后来改变了寻址方式，通过使用多个寄存器对同一状态不同信息的储存和判断，使字符能够一个像素点一个像素点的下落，从而使字符能够平滑落下。
- **字符消失不正常的问题：**当同一列产生两个字符时，一个字符掉到最底下消失的时候会连带着上面的字符一起消失。增加了一个valid位，改变了显示方式，如果valid位=1即该列不生成新的，如果valid位=0，即将生成的放在该列上，保证每列每次只会生成一个，这样就不会出现同时消失的问题。
- **键码问题：**一开始判断一直有问题，会出现一下子消掉很多个，才发现是通码和断码的问题，后来增加了一个flag_word位，对PS2_KEYBOARD模块进行稍微的修改，即实现按下去一次消一个的功能。

## 十、 实验分工

朱嘉琦：键盘控制模块，随机数生成模块，字符掉落等功能

王珺珧：基本vga框架代码，字符掉落显示模块等功能

## 十一、 实验反馈

感谢所有老师和助教的辛勤付出！