实验名称：实验十一 字符输入界面

姓名：张涵之

学号：191220154

班级：周一 5-6

邮箱：191220154@smail.nju.edu.cn

实验时间：2020/11/30

11.3 实验内容

实现一个可以用键盘输入，并在 VGA 显示器上回显的交互界面。

基本要求：支持所有小写英文字母和数字、符号输入；

一直按压某个键时，重复输出该字符；

输入至行尾后自动换行，输入回车也换行；

扩展要求：支持 BackSpace 键删除字符；

删除至本行开始后，再按删除回车键，停留在上一行末尾的非空字符后；

支持 Shift 键、Caps 键以及大小写字符输入。

实验目的：实现一个可以用键盘输入，并在 VGA 显示器上回显的交互界面。

实验原理：

1）字符显示：ASCII 字符用 7bit 表示，共 128 个字符。大部分情况下用 8bit 来表示单个字符，系统预留 256 个字符。系统中预先存储这 256 个字符的字模点阵。

图 11-1: ASCII 字符字模

这里每个字符高为 16 个点，宽为 9 个点。单个字符可以用 16 个 9bit 数来表示，每个数代表字符的一行，对应的点为“1”时显示白色，为“0”时显示黑色。只需要 256 × 16 × 9 ≈ 37kbit 的空间即可存储整个点阵。在显示时，根据当前屏幕位置，确定应该显示哪个字符，再查找对应的字符点阵即可完成显示。对于 640×480 的屏幕，可以显示 30 行 (30×16=480)，70 列（70 × 9=630)。系统的显存需要 30 × 70 大小，每单元存储 8bit 的 ASCII 字符。

2）扫描显示：VGA 控制模块可以输出当前扫描到的行和列的位置信息，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)。利用坐标查询字符显存，获取对应字符的 ASCII 编码。利用 ASCII 编码查询对应的点阵 ROM，根据扫描线的行和列信息，知道当前扫描的是字符内哪个点，根据该点对应的 bit 选择输出颜色。

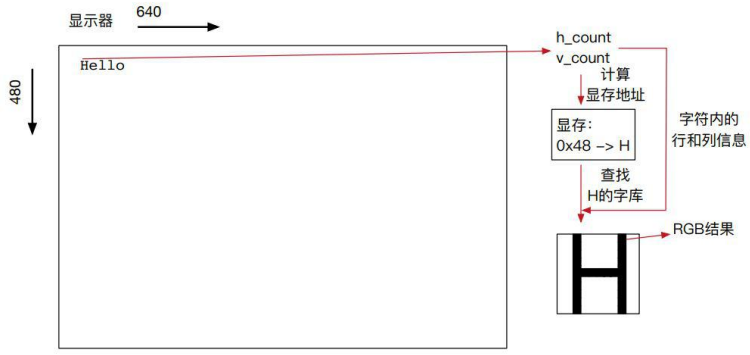
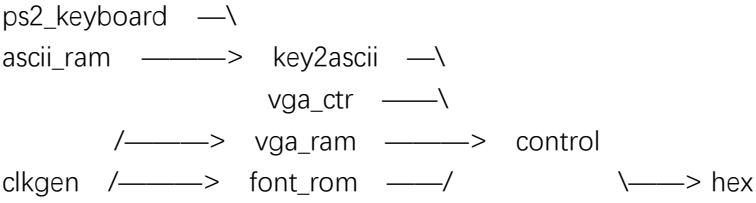


图 11-3: 字符显示流程示意图

显示的过程总结如下：
1. 根据当前扫描位置获取对应字符的 x，y 坐标，以及扫描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，获取对应的 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。

3）显存读写：对于键盘输入可以复用之前实现的键盘控制器。在键盘有输入的时候对字符显存进行改写，将按键对应的 ASCII 码写入显存的合适位置，将输入直接反馈到屏幕上。

程序代码或流程图：



ps2_keyboard　　//键盘控制器获取键码
ascii_ram　　//读文件将键码转化为对应的 ascii 码
*以上两个模块基本复用 exp08 中代码

key2ascii　　//综合键盘控制器和 ram 并判断 shift 和 caps 键

```
ps2_keyboard p(clk,clr,ps2_clk,ps2_dat,code,ready,nextdata,overflow);
ascii_ram a(scancode,caps,ascii);

always @ (posedge clk) begin
   if (ready && nextdata) begin
      nextdata <= 0;
      scancode <= code;
      if (code == 8'h12 || code == 8'h59) caps <= ~caps;
      if (code == 8'hf0) begin
         times <= times + 1;
         flag <= 1;
      end
      else begin
         if (flag) begin
            en <= 0;
            flag <= 0;
            if (code == 8'h58) caps <= ~caps;
         end
         else en <= 1;
      end
   end
   else nextdata <= 1;
end
```

其中 shift 按下和弹起时 caps 各取反一次，caps 按下弹起一次 caps 才取反

vga_ctrl　　//显示器控制器提供接口
clkgen　　//生成特定频率的时钟
*以上两个模块基本复用 exp09 中代码

vga_ram　　//存储和读写屏幕上 70 × 30 的内容
font_rom　　//存储 ascii 码黑白点阵图
hex　　//将 ascii 码显示在七段数码管上便于调试

control　　//综合键盘和显示器的顶层控制模块

```
module control(
    input clk,                  //系统时钟
    input clr,                  //清零
    input ps2_clk,
    input ps2_dat,              //键盘模块接口
    output vga_clk,
    output valid,
    output hsync,
    output vsync,
    output [7:0] vga_r,
    output [7:0] vga_g,
    output [7:0] vga_b,         //显示器模块接口
    output [6:0] hex0,
    output [6:0] hex1           //七段数码管
);

    wire [23:0] vga;            //最终颜色数据
    wire [9:0] h_addr;
    wire [9:0] v_addr;          //屏幕坐标
    wire [8:0] font;            //当前字符行点阵
    wire [7:0] vga_addr;        //vga_ram读出数据
    wire [7:0] ascii;           //当前按键的ascii码
    wire [7:0] ascii_out;       //键盘控制器输出值
    wire [7:0] waste;           //没什么用的虚假寄存器
    wire en;                    //键盘使能输出
    wire keyboard_clk;          //键盘时钟

    reg [23:0] data = 24'hffffff;     //过程中的颜色数据
    reg [11:0] block_addr = 0;        //vga_ram读出地址
    reg [11:0] index = 0;             //vga_ram写入地址
    reg [11:0] ram_index [29:0];      //用于删除的旧坐标寄存器
    reg [11:0] address;               //font_rom读出地址
    reg [11:0] preindex;              //用于删除的旧坐标变量
    reg [7:0] ascii_data;             //vga_ram写入数据
    reg [3:0] times = 0;              //键盘用计数器
    reg flag = 1;                     //判断长按的标志
    reg backspace_state = 1;          //删除状态的标志
    reg enter_state = 1;              //回车换行的标志

    assign vga_r = vga[23:16];              //红
    assign vga_g = vga[15:8];               //绿
    assign vga_b = vga[7:0];                //蓝
    assign ascii = ascii_out & {8{en}};     //按键时有效

    clkgen #25000000 clk_v(clk,1'b0,1'b1,vga_clk);
    clkgen #20 clk_k(clk,1'b0,1'b1,keyboard_clk);                          //生成两种时钟
    key2ascii k(clk,clr,ps2_clk,ps2_dat,en,ascii_out);                     //键盘转ascii码
    vga_ctrl v_C(vga_clk,1'b0,data,h_addr,v_addr,hsync,vsync,valid,
       vga[23:16],vga[15:8],vga[7:0]);                                     //显示器控制
    font_rom f_r(.address(address),.clock(vga_clk),.q(font));              //字符点阵读取
    vga_ram v_r(block_addr,index,vga_clk,keyboard_clk,1'b0,ascii_data,
       1'b0,1'b1,vga_addr,waste);                                          //屏幕信息存取

    hex h0(ascii[3:0],en,hex0);
    hex h1(ascii[7:4],en,hex1);                                            //七段数码管调试
```

//计算两个存储器中读取的坐标，并根据当前点阵设置颜色

```
    always @ (posedge vga_clk) begin
       block_addr = (v_addr >> 4) * 70 + (h_addr / 9);
       address = (vga_addr << 4) + (v_addr % 16);
       if (h_addr < 8) data = 24'h0000000;
       else if (font[(h_addr + 2) % 9] == 1'b1) data = 24'hffffff;
       else data = 24'h000000;
    end
```

//记录光标上一次的位置，用于删除时退回

```
    always @ (negedge keyboard_clk) begin
       if (index < 70) preindex <= 0;
       else preindex <= ram_index[index / 70 - 1];
    end
```

//按键时：进入退格键单个删除模式

```
    always @ (negedge keyboard_clk) begin
        if (ascii != 8'h0) begin
            if (times == 0) begin
                if (ascii == 8'h0d) begin
                    backspace_state <= 0;
                    ram_index[index / 70] <= index + 1;
                    index <= index - (index % 70) + 70;
                    enter_state <= 0;
                end
```

//进入回车键单个换行模式

```
                else if (ascii == 8'h08) begin
                    if (backspace_state == 0) begin
                        index <= index;
                        backspace_state <= 1'b1;
                    end
                    else begin
                        if (index % 70 == 0) index <= preindex;
                        else index <= index - 1;
                    end
                end
```

//进入其他按键单个输入模式

```
                else begin
                    if (backspace_state || !enter_state) index <= index;
                    else begin
                        if ((index + 1) % 70 == 0) ram_index[(index - 1) / 70] <= index;
                        index <= index + 1;
                    end
                    backspace_state <= 0;
                    enter_state <= 1;
                end
                times <= times + 1;
            end
```

//按键超过十个周期，进入连续删除/换行/输入模式

```
            else if (times == 4'd10) begin
                if (ascii == 8'h0d) index <= index - (index % 70) + 69;
                else if (ascii == 8'h08) begin
                    if(index % 70 == 0) index <= preindex;
                    else index <= index - 1;
                end
                else begin
                    if ((index + 1) % 70 == 0) ram_index[(index - 1) / 70] <= index;
                    index <= index + 1;
                end
            end
            else begin
                index <= index;
                times <= times + 1;
            end
            ascii_data <= ascii;
        end
```

//没有按键时：光标保持，计数器清零，超出屏幕范围归零

```
        else begin
            index <= index;
            times <= 0;
        end
        if(index >= 2100) index <= 0;
    end

endmodule
```

实验环境/器材：实验箱一个，笔记本电脑一台，键盘一个，显示器一个。

实验步骤/过程：

分开编写各模块的代码，用七段数码管显示参数，分别进行调试。

将提供的 ascii 码点阵文本文档编成 mif 文件，用 IP 核生成单口 rom 存储。

用 IP 核生成双口 ram 读写（存取）屏幕上的坐标信息。

对键盘和显示器进行调试，观察显示器的输出，对代码进行修改。

测试方法：按下键盘，在显示器上显示字符。
实验结果：字符的显示符合预期。

实验中遇到的问题及解决办法：
时序配合不当，键盘输入在七段数码管上可以正确显示，但显示器没有输出。
修改时序逻辑，增加不同的时钟分别用于键盘和显示器存储的读写。

屏幕显示的字符可以看出符合输入，但看起来边缘总有 1-2 个像素的错位。
每行最前面输入第一个字符时，会显示两次，其余字符正常。
思考应该是各模块时序配合仍然不够好，然而这种程度的小错误似乎并不需要也不值得对整体框架进行重构，于是决定人为地对下标进行判断，将每行最前面的重复字符的第一个置为空白，加上一个通过观察和调试得出的常数来修正相位差，得到了看似正常的显示效果。

实验得到的启示：无。

意见和建议：无。