实验报告lab09

姓名 林凡琪

学号 211240042

班级 匡亚明学院

邮箱 211240042@smail.nju.edu.cn

实验时间 2022/12

实验目的

本实验将利用前面实现过的键盘和显示器功能来搭建一个简单的字符输入界面，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。

实验原理（知识背景）

- 字符显示

  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。

  有了字符点阵后，系统就不再需要记录屏幕上每个点的颜色信息了，只需要记录屏幕上显示的ASCII 字符即可。在显示时，根据当前屏幕位置，确定应该显示那个字符，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）的ASCII 字符。系统的显存只需要30 × 70 大小，每单元存储8bit 的ASCII 字符即可。这样，我们的字符显存只需要2.1kByte，加上点阵的6.144kByte，总共只需要不到10kByte的存储，FPGA 片上的存储足够实现了。

- 系统设计

- 扫描显示
- 显存读写

实验环境/器材等

Verilog 2022.1

Windows 10

Nexys开发板

程序代码或流程图

```
54   //keyboard variables
55   reg nextdata_n;
56   wire [5:0] dp;
57   wire ready;
58   wire [7:0] keydata;
59
60   wire[7:0] hexbuf0,hexbuf1,hexbuf2,hexbuf3;
61
62   reg [7:0] key_count;
63   reg pressing;
64   reg ignore_next;
65   reg [7:0] current_key;
66   wire [7:0] ascii_key;
67
68   //VGA part
69
70   wire [9:0] h_addr;
71   wire [9:0] v_addr;
72   wire [23:0] vga_data;
73
74   wire [6:0] h_char; //char 70 per line
75   wire [4:0] v_char; //char 30 lines
76   wire [3:0] h_font; //font 9 points horizontal
77   wire [3:0] v_font; //font 16 point vertical
78
79   reg [7:0] m_char; //current character
80
81
82
83   reg [6:0] h_cur;
84   reg [4:0] v_cur;
85       |   |   new_key;
86   reg [1:0] in_state; //to wait for scancode translate
87   wire      cursor;
88   wire      clk_cur;
89   reg       char_wr;
90   wire [11:0] char_addr;
91   reg [11:0] char_wr_addr;
92   wire [11:0] char_rd_addr;
93   reg [7:0] char_buf_data;
94   wire [7:0] char_out;
95   reg [6:0] line_end[31:0];
96   reg [4:0] line_offset;
97   reg [6:0] clear_point;
98
99   ////////////////////////////
100      structural coding
```

```
104   ps2_keyboard mykey(CLOCK_50, KEY[0], PS2_CLK, PS2_DAT, keydata, ready,
      nextdata_n,LEDR[])
105
106   bcd7seg seg0(cerrent_key[3:0],hexbuf0);
107   bcd7seg seg1(cerrent_key[7:4],hexbuf1);
108   bcd7seg seg2(ascii_key[3:0],hexbuf2);
109   bcd7seg seg3(ascii_key[7:4],hexbuf3);
110
111
112
113   bcd7seg seg4(key_count[3:0],{dp[0], HEX4[6:0]});
114   bcd7seg seg5(key_count[7:4],{dp[0], HEX5[6:0]});
115
116   scancode_ram myram(CLOCK_50,current_key, ascii_key);
117
118   assign HEX0=pressing? hexbuf0[6:0]: 7'h7f;
119   assign HEX1=pressing? hexbuf1[6:0]: 7'h7f;
120   assign HEX2=pressing? hexbuf2[6:0]: 7'h7f;
121   assign HEX3=pressing? hexbuf3[6:0]: 7'h7f;
122
123   //assign LEDR[8]=ready;
124   //assign LEDR[7:0] = keydata;
125
126   always @(posedge CLOCK_50)
127   begin
128       if(ready==1'b1 && nextdata_n==1'b1)
129       begin
130
131           /*test code
132           scancide0<=keydata;
133           scancode1<=scancode0;
134           scancode2<=scancode1;*/
135           if (keydata==8'hF0)//break code
136           begin
137               ignore_next<=1'b1;
138               pressing<=1'b0;
139               current_key<=8'b0;
140           end
141           else if (keydata==8'E0)
142           begin
143               ignore_next <= 1'b1;
144           end
145           else if (ignore_next)
146           begin
147               ignore_next<=1'b0;
148           end
149           else //nomal key
```

```
177     always @(posedge CLOCK_50)
178     begin
179         if(KEY[0] == 1'b0)//reset
180             begin
181                 h_cur <= 7'h0;
182                 v_cur <= 6'h0;
183                 in_state <= 2'd0;
184                 line_offset <= 5'd0;
185                 clear_point = 7'd0;
186                 char_buf_data <= 8'h00;
187                 char_wr_addr <= {clear_point, 5'd31};
188             end
189         else
190         begin
191             case(in_state)
192             2'd0: begin
193                     if(new_key==1'b1) in_state <= 2'd3;
194                     char_wr <= 1'b0|~VGA_CLK;
195                     if(~char_wr) //not writing
196                     begin
197                         char_buf_data <= 8'h00;//clean the unused lines
198                         clear_point <= clear_point + 1'd1;
199                         char_wr_addr <= {clear_point, (5'd31+line_offset)};
200                         line_end[5'd31+line_offset]=7'd0;
201                     end
202                 end
203             2'd1: begin
204                     in_state <= 2'd0;
205                     if(current_key == 8'h66) //backspace
206                     begin
207                         char_buf_data <= 8'h00;
208                         char_wr <= 1'b1;
209                         if(h_cur == 7'd0)
210                         begin
211                             if(v_cur==5'd0)
212                             begin
213                                 h_cur<=7'd0;
214                                 v_cur<=5'd0;
215                                 char_wr_addr <= 12'd0+{7'd0, line_offset};
216                             end
217                             else
218                             begin
219                                 h_cur <= line_end[v_cur-5'd1+line_offset];
220                                 v_cur <= v_cur - 5'd1;
221                                 char_wr_addr <= {7'd69, (v_cur - 5'd1
                                    +line_offset)};
222                             end
223                         end
```

实验步骤/过程

根据实验要求，本次实验是将前面所完成的lab7和lab8结合起来进行的，旨在综合运用之前所学知识，深入理解相关概念并加深印象。因此，本次实验中除了复用前面的代码，还需要进一步完善已有代码的功能，并在此基础上添加新的代码，以将键盘和VGA这两个板块合在一起。具体而言，需要进行以下方面的工作：

- 对lab7和lab8的代码进行整合和优化，以确保代码的稳定性和可读性。

- 在已有的代码基础上，添加新的功能模块，例如对用户输入的数据进行处理和验证，以及生成相应的输出结果。
- 进一步调试和测试代码，确保其能够正确地运行并满足实验要求。
- 进行引脚连接，编写缺省文件。

测试方法

上板检测，连接了键盘和VGA。

实验结果

实现了一个可以用键盘输入，并在VGA 显示器上回显的交互界面。

完成了基本要求：

- 支持所有小写英文字母和数字输入，以及不用Shift 即可输入的符号。
- 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。

完成了拓展要求：

- 可以显示光标
- 支持BackSpace 键删除光标前的字符。
- BackSpace 删除至本行开始后，再按BackSpace 可以删除回车键，光标停留在上一行末尾的非空字符后。
- 支持自动滚屏，即输入到最后一行后回车出现新空白行，并且所有已输入的行自动上移一行。
- 支持Shift 键以及大小写字符输入。

实验中遇到的问题及解决办法

问题：因为写了很多版，所以很多文件经常路径不对。

解决：仔细核对每个文件的路径位置并进行改正。

实验得到的启示

通过本次实验，更深入地了解代码的开发过程和调试方法，同时也可以提高自己的问题解决能力和编程技巧。

意见和建议