2020-11-29

数字电路实验十一 字符输入界面

秦嘉余

191220088

1348288404@qq.com

目录

1.实验目的

本实验的主要目的是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。

2.实验原理

1.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，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 $256 \times 16 \times 9 \approx 37$kbit 的空间即可存储整个点阵。

2.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($0 \leqslant x \leqslant 69$, $0 \leqslant y \leqslant 29$)。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。

3.显存读写

对于键盘输入，我们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，这样输入就可以直接反馈到屏幕上了。

3.实验环境/器材

本次实验的环境为 Quartus17.1 版本
本次实验的器材为 DE10 Standard 开发板

4. 显示器交互界面

1. 设计思路

本次实验主要是对之前模块的复用，在对于交互方面的代码进行重写。由于显示字符有大量的数据需要存储，所以此次实验需要多个 rom 或者 ram 来存储各种数据，包括界面上各个位置字符的 ascii 码数据，字符的显示数据，以及 freq 频率的数据等等。之后，复用原来的键盘代码改进，根据键码向对银行 rom 中写入数据，然后显示屏不断读取 rom 中的数据，根据 ascii 字符以及行列数显示黑白的像素点即可。

2. 实验代码

顶层模块如下，与之前的实验 9 代码基本相同，主要的修改集中于 getcolor 模块中。

```
wire [9:0] h_addr,v_addr;
wire [23:0] vga_data;
assign VGA_SYNC_N=0;


//=======================================================
//  Structural coding
//=======================================================
   clkgen #(25000000) my_vgaclk(CLOCK2_50,1'b0,1'b1,VGA_CLK);
   vga_ctrl my_ctrl(VGA_CLK,1'b0,vga_data,h_addr,v_addr,VGA_HS,VGA_VS,VGA_BLANK_N,VGA_R,VGA_G,VGA_B);
   getcolor(CLOCK2_50,VGA_CLK,PS2_CLK,PS2_DAT,h_addr,v_addr,vga_data,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,LEDR[0]);

endmodule
```

对于 getcolor 模块，声明的 rom，ram 如下：a 为键盘键码与 ascii 码的映射 ram，与 ps2_keyboard 配合使用，display 模块为显示 rom，记录显示屏上每个位置的字符的 ascii 码信息，font_data 模块为每个字符显示的颜色 rom，是显示屏上每个像素的显示需要的信息

```
(* ram_init_file = "ascii.mif" *)reg [7:0] a [255:0];
ps2_keyboard i(.clk(vga_clk),.clrn(1'b1),.ps2_clk(ps2_clk),.ps2_data(ps2_data),.data(data),.nextdata_n(1'b0),.ready(ready));
display d(vga_clk,code,addr,idx,we,q);
font_data fram(line,vga_clk,0,0,font);
```

getcolor 的接口如下：

```
module getcolor(clk,vga_clk,ps2_clk,ps2_data,h_addr,v_addr,vga_data,HEX0,HEX1,HEX2,HEX3,HEX4,HEX5,blank);

   input clk;
   input vga_clk;
   input ps2_clk,ps2_data;
   input [9:0] h_addr;
   input [9:0] v_addr;
   output reg [23:0] vga_data;
   output reg [6:0] HEX0,HEX1,HEX2,HEX3,HEX4,HEX5;
   //reg [11:0] ram [4095:0];
   reg [11:0] addr;
   reg [11:0] line;
   //reg [7:0] ascii [2099:0];
   //reg [11:0] data;
   output reg blank;
   wire ready;
   wire [7:0] data;
   reg [11:0] cnt;
   reg [11:0] idx;
   reg break;
   reg we;
   wire [7:0] q;
   reg [7:0] code;
   wire [10:0] font;
```

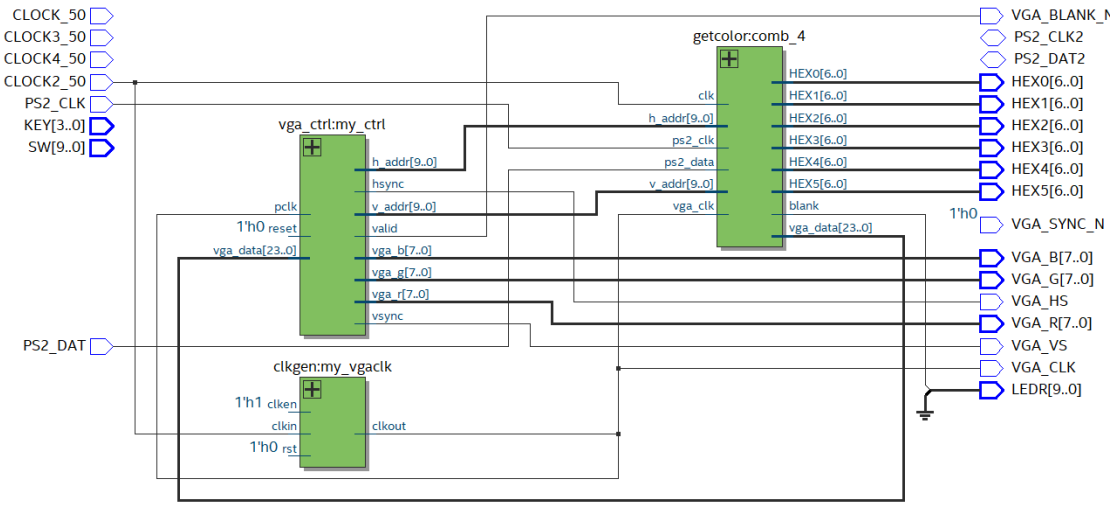
程序利用如下代码计算对应点的像素信息，传给显示屏显示，其中 blank 为当前像素前的明暗信息，addr 为当前的 ascii 字符索引，line 为当前显示的 ascii 字符的显示数据的当前行

```
always @(vga_clk)
begin
   addr<=(h_addr/9)+(v_addr>>4)*70;
   line<=v_addr[3:0]+(q<<4);
end


always @(h_addr or v_addr)
begin
   blank<=font[(h_addr+7)%9];
end


decode_hex hex0(code[3:0],HEX0);        //why?
decode_hex hex1(code[7:4],HEX1);
decode_hex hex2(idx[3:0],HEX2);
decode_hex hex3(idx[7:4],HEX3);
decode_hex hex4(cnt[3:0],HEX4);
decode_hex hex5(cnt[7:4],HEX5);


always @(h_addr or v_addr)        //clk??
begin
   if(blank && (h_addr<630))
      vga_data<=24'hffffff;
   else
      vga_data<=24'h000000;
end
```

we 为 display 这个 rom 的读写信号，idx 与 cnt 为分别为当前输入位置的索引与下一个需要输入的位置的索引

```
always @ (posedge vga_clk)
begin
    if (data == 8'hf0)
    begin
        break<=1'b1;
    end
    else if (break && data != 0)
    begin
        break<=1'b0;
    end
    else
    begin
        if(data!=0)
        begin
            if(data==8'h5a)
            begin
                idx<=cnt;
                cnt<=(cnt+70)/70*70;
                we<=0;
            end
            else
            begin
                we<=1;
                idx<=cnt;
                cnt<=cnt+1;
                //if(caps&&a[data]>=8'h61&&a[data]<=8'h7a)
                // code<=a[data]-8'h20;
                //else
                code<=a[data];
            end
        end
    else
```

3.RTL 视图



4.引脚分配

该实验为 system 软件自动分配的引脚，所以不再展示、

5.实验结果

本实验已在开发板上验收

5.实验实验中遇到的问题及解决办法

问题一：
实验中，显示的时序逻辑十分重要，如果读写的顺序不合适，会导致输入输出不同步，从而显示出现故障，在此次实验中，选择 vga 的时钟比较合适，时钟速度较为合理
问题二：
由于在本次实验中，各种位置信息的计算，以及 rom 的读写都有一定的时间消耗，所以在显示字符时会导致字符的显示出现偏移现象，这种现象由于本身时序逻辑的复杂性较难消除，为了解决这个问题，在显示字符时，加上一个偏移量读取像素点的明暗信息，抵消这一偏移，可以完美的显示字符
问题三：
本次实验中很多 rom 的内容量较大，不使用 IP 核来生成 rom，直接使用寄存器来读取，会发生无法写入读取的问题，换用合适的 rom 即可解决这一问题