数字电路与数字系统

实验报告

实　　验: 字符输入界面

姓　名: 周心同

学　　号: 201220069

目录

1.实验目的

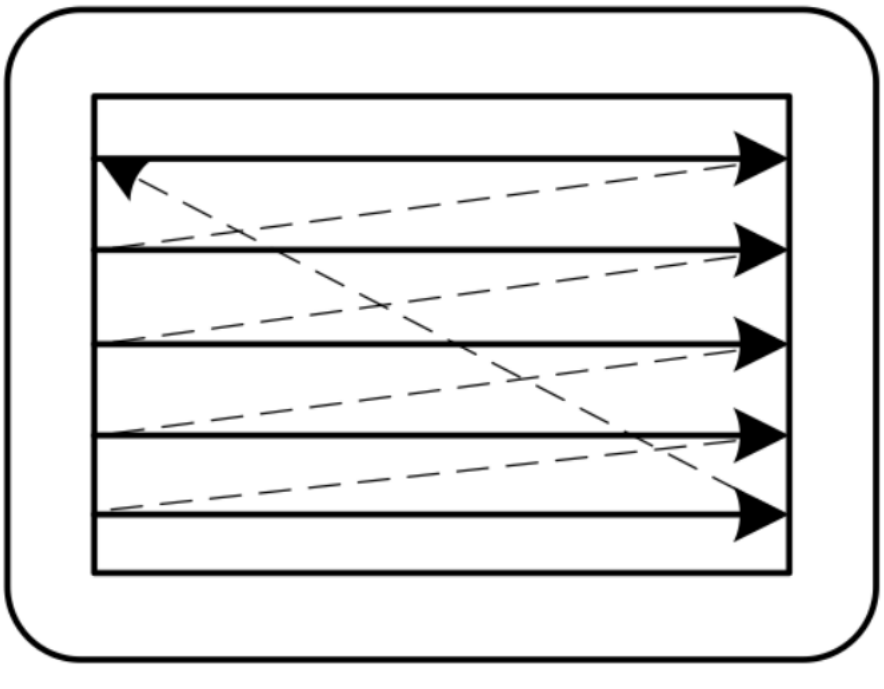
搭建一个简单的字符输入界面

深入理解多个模块之间的交互和接口的设计

2.实验原理

2.1 VGA

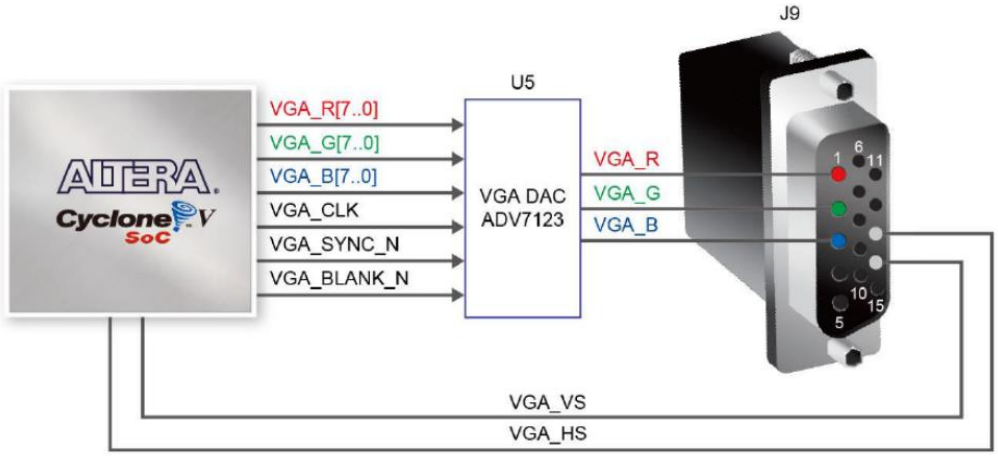
工作原理：VGA 每一帧图像的显示都是从屏幕的左上角开始一行一行进行的，行同步信号是一个负脉冲，行同步信号有效后，由 RGB 端送出当前行显示的各像素点的 RGB 电压值，当一帧显示结束后，由帧同步信号送出一个负脉冲，重新开始从屏幕\的左上端开始显示下一帧图像，如图所示。



RGB 端并不是所有时间都在传送像素信息，由于 CRT 的电子束从上一行 的行尾到下一行的行头需要时间，从屏幕的右下角回到左上角开始下一帧也需 要时间，这

时 RGB 送的电压值为 0（黑色），这些时间称为电子束的行消隐时 间和场消隐时间，行消隐时间以像素为单位，帧消隐时间以行为单位。

显示原理：DE10-Standard 开发板上使用了一块 VGA DAC ADV7123 芯片来实现 VGA 功能。该芯片完成 FPGA 数字信号到 VGA 模拟信号的转换，具体连接方式如 图 所示。



开发板和 ADV7123 芯片之间的接口引脚包括 3 组 8bit 的颜色信号 VGA_R[7:0], VGA_G[7:0], VGA_B[7:0]，行同步信号 VGA_HS，帧同步信号 VGA_VS，VGA 时钟信号 VGA_CLK，VGA 同步（低有效）VGA_SYNC_N, 和 VGA 消隐信号（低有效）VGA_BLANK_N。

2.2 字符显示

上面已经说明了怎么在 vga 上显示，接下来就是显示字符的问题。

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。因而，我们可以设计一个点阵文本文件，通过该文件将传入字符的 ascii 码转换为要显示的内容。这里老师已经给出，其中每 3 个 16 进制数 (共 12bit) 表示单个字符的一行，该行的 9 个点中的最左边点在 12bit 中的最低位（请注意高低位顺序），然后依次类推，最高的 3 个 bit 始终为 0。每个字符 16 行，共 256 个字符。

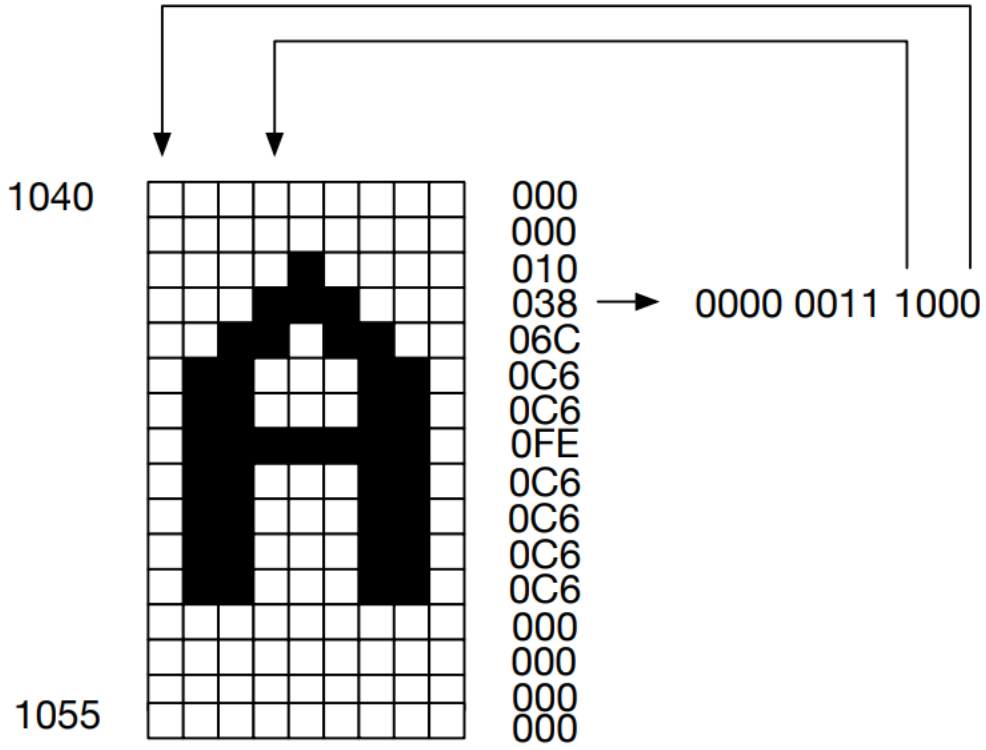


图 9-2: 字模 “A” 与存储器的关系

这样在显示时，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。

2.3 显示位置

首先，需要让 vga 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。

2.4 显存

由于键盘时钟和 vga 时钟不一样，所以显存要有两个端口，一个用于读取键盘输入，一个用于 vga 扫描键盘输出。

3.实验环境

- 软件环境

Quartus 17.1 Lite

- 硬件环境

开发板：DE10 Standard

FPGA：Intel Cyclone V SE 5CSXFC6D6F31C6N

4.实验步骤和结果

4.1 显存模块

```
                                wren_b,
                    ram_data,
                                now_ascii);
//vga根据block_addr访问显存取出ram_data
//键盘输入 来改变应当显示的值
//这个模块作用是避免因两个时钟频率不一致而带来的读写时序问题
   reg [7:0] regs[4095:0];

   input [11:0]  ram_out_addr;
   input [11:0]  ram_in_addr;
   input   VGA_CLK;
   input   PS2_CLK;
   input [7:0]  data_a;
   input [7:0]  ascii;
   input   wren_a;
   input   wren_b;
   output   reg [7:0]  ram_data;
   output   [7:0]  now_ascii=0;

   reg [7:0] preascii;
   integer flag1=0;
   integer flag2=0;

   reg prewren;

always @(posedge VGA_CLK)
begin
     ram_data<= regs[ram_out_addr];
end

always @(posedge PS2_CLK)
begin

    if(wren_b && ascii!=8'h0d && ascii!=8'h08 && ascii!=8'hff && ascii!=8'hfe && ascii!=8'hfd && ascii!=
    regs[ram_in_addr]<=ascii;

end



endmodule
```

两个时钟读写，避免因两个时钟频率不一致而带来的读写时序问题。

之所以不用 ip 核是因为要控制回车 退格等特殊符号不输入。

4.2 vga 驱动信号模块和 keyboard 键盘输入模块

与实验七，实验八基本相同，不同的是添加了拓展功能，详细就在后文叙述。

4.3 顶层模块

这里顶层模块要干的事情很多。

首先是接受到键盘输入的 ascii 码:

```
keyboard k(CLOCK_50,
    1,
    PS2_CLK,
    PS2_DAT,
    a,//计数
    c,//扫描码
    ascii_keyboard,//ascii
    LEDR[0],//ctrl
    LEDR[1],//shift
    flag,
    LEDR[2]//caps
);
 //记录输入的字符 输出ascii码
```

因为一次输出涉及到 11 个时钟周期，所以要加个计数器，以确保按一次键不会跳出来 11 个字母。

```
always @(posedge PS2_CLK)
begin
    if(cnt==10)
   begin
      cnt<=0;
      ascii<=ascii_keyboard;
   end
   else
   begin
      cnt<=cnt+1;
      ascii<=0;
   end
end
```

然后对接显存模块，将 ascii 码写入，将扫描的取出。

```
keyboard_ram_vga rv(ram_out_addr,
                            ram_in_addr,
                    VGA_CLK,
                            PS2_CLK,
                     1'b0,
                            ascii,
                    1'b0,
                            writein,
                   ram_data,
                            now_ascii);
 //vga根据block_addr访问显存取出ram_data
 //键盘输入 来改变应当显示的值
 //这个模块作用是避免因两个时钟频率不一致而带来的读写时序问题
```

根据扫描位置，从对应的显存位置中取出 ascii 码，然后根据 ascii 码在 font 中的位置并结合显存模块读出来应该显示的颜色。

```
always @(VGA_CLK)begin
    ram_out_addr <= (v_addr >> 4) * 71 + ((h_addr+2) / 9);//算出ram中取哪一行

    address <= (ram_data << 4) + (v_addr % 16);//以取出的ascii码为索引，找到字符表中的位置
    if(font_data[h_addr % 9] == 1'b1)
      vga_data <= 24'hffffff;
    else
      vga_data <= 24'h000000;
end   //
```

对接 font 模块

```
font (
  address,
  VGA_CLK,
  font_data);
```

4.4 写入位置改变及退格回车方向键的拓展功能

```
always @ (posedge PS2_CLK)//写入地址改变
   begin
       if(!flag2)
       begin
       for(i=0;i<31;i=i+1)
         ram_index[i]=4095;//初始值
       flag2=1;

       end

      if(ascii != 0)
      begin
         if(ascii == 8'h0d) //回车
            begin
               ram_index[ram_in_addr/71]<=ram_in_addr;
               ram_in_addr <= ram_in_addr + 71 - (ram_in_addr % 71);

               writein<=1;
            end
         else if (ascii == 8'h08 )//退格
            begin
               if(ram_in_addr % 71 !=0)
               begin
               ram_in_addr <= ram_in_addr-1;
                 if (ram_index[ram_in_addr/71]== ram_in_addr)
                 ram_index[ram_in_addr/71]<=ram_in_addr-1;
               end
               else//删除回车 退回上一行
                  begin
                     if(ram_in_addr < 71 )
                     begin
                     ram_in_addr<=0;
                     ram_index[0]=0;
                     end
                     else
                     begin
                     ram_in_addr <= ram_index[ram_in_addr/71-1];
```

```
                    begin
                    ram_in_addr <= ram_index[ram_in_addr/71-1];
                    ram_index[ram_in_addr/71]<=4095;
                    end
                end
            writein<=1;
        end
    else if(ascii == 8'hff )//左
        begin
        if(ram_in_addr % 71 !=0)//不在最左边
          ram_in_addr <= ram_in_addr-1;
          else//在最左边
              begin
                  if(ram_in_addr < 71 ) //代表为0
                  ram_in_addr<=0;
                  else
                  ram_in_addr <= ram_index[ram_in_addr/71-1];
              end
          writein<=0;
        end
    else if(ascii == 8'hfe )//下
```

```
                    ram_in_addr<=0;
                    else
                    ram_in_addr <= ram_index[ram_in_addr/71-1];
                end
            writein<=0;
        end
    else if(ascii == 8'hfe )//下
        begin
          if(ram_index[ram_in_addr/71+1]!=4095 && ram_index[ram_in_addr/71+1]%71>=ram_in_addr%71)
          ram_in_addr<=ram_in_addr+71;
          else if(ram_index[ram_in_addr/71+1]!=4095)
          ram_in_addr<=ram_index[ram_in_addr/71+1];

          writein<=0;
        end
    else if(ascii == 8'hfd) //右
        begin
          if(ram_index[ram_in_addr/71]!=4095 && ram_index[ram_in_addr/71]>ram_in_addr)
          ram_in_addr<=ram_in_addr+1;
          else if(ram_index[ram_in_addr/71+1]!=4095)
          ram_in_addr<=ram_in_addr + 71 - (ram_in_addr % 71);

          writein<=0;
        end
    else if(ascii== 8'hfc) //上
        begin
          if(ram_in_addr >=71)
          if (ram_index[ram_in_addr/71-1] % 71 >=ram_in_addr %71)
          ram_in_addr<=ram_in_addr-71;
          else
          ram_in_addr<=ram_index[ram_in_addr/71-1];
          writein<=0;
        end
    else
       begin
          if((ram_in_addr + 1) % 71 == 0) ram_index[ram_in_addr/71] <= ram_in_addr;
          else ram_index[ram_in_addr/71]<=ram_in_addr+1;
          ram_in_addr <= ram_in_addr + 1;

          writein<=1;
        end
  end

  if(ram_in_addr == 2130) ram_in_addr <= 0;
end
```

对于基础功能而言，主要实现两个：每次键盘输入如果是回车，就移到下一行的首地址，否则就往后移一个位置。

对于拓展功能而言，实现内容就很多：首先，要有一个数组存储每行的最后一个的字符的位置，在执行各个操作时注意存储，同时这里我初始化为 4095。对于退格键，如果不是行首就往前一格，否则退到上行最后一个字符。对于方向键，需要考虑各种情况，以最麻烦的右为例，如果右边有字符（即当前地址不是改行的最后一个），就往右边一格，如果没有，看下行有无字符，有就到下行，没有就不动，注意在这里如果回车了该行数组值是 0 而不是 4095，将被视为有换行符，所以不可能跨行有字符，同时退格消除回车符时会改回 4095 无字符状态。

4.5 shift 键和大小写字符输入

这里很简单，因为照搬了实验七的键盘模块，略微改一下 shift 情况下特殊字符的输出就行。

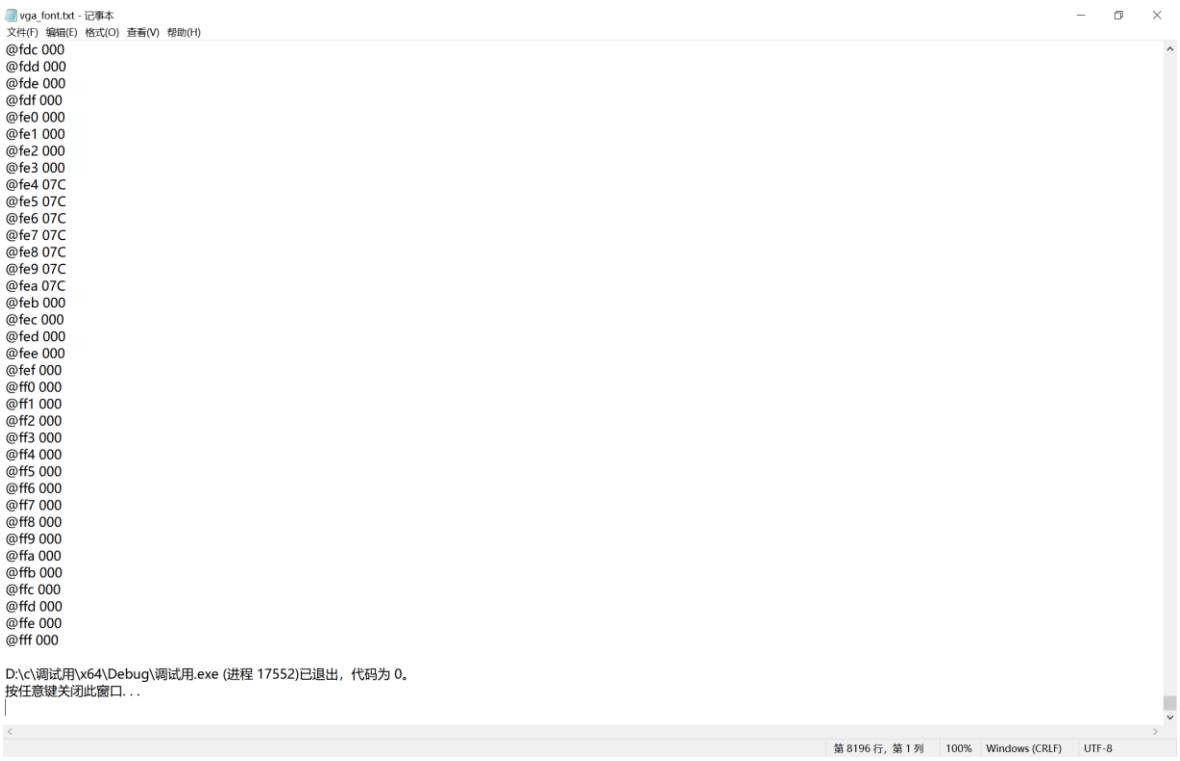
4.6 光标和闪烁

```
integer count=0;
reg [23:0] blink=0;
always @(posedge pclk)
begin
    if(count == 2499999*4+1)
    begin
       count<=0;
       blink=8'hffffffff-blink;
    end
    else
       count<=count+1;


end


 always @(posedge pclk)
 begin
      if((v_addr >> 4) * 71 + ((h_addr) / 9)==ram_in_addr && blink==8'hffffffff)
        begin
            vga_r = blink[23:16];
            vga_g = blink[15:8];
            vga_b = blink[7:0];
        end
      else
        begin
            vga_r = vga_data[23:16];
            vga_g = vga_data[15:8];
            vga_b = vga_data[7:0];
        end
 end
```

在扫描到的位置为当前键盘要输入的位置时，输出一个 blink，而 blink 是一个随时钟闪烁的值就行。

4.7 font

这个模块有点特殊，因为不会 txt 转 mif，于是再一次用到了 c 语言。



然后用 readmemh 读入

```
module font (
    address,
    clock,
    q);
input [11:0] address;
input clock;
output [11:0] q;

reg [11:0] regs [4095:0];
initial
begin
$readmemh("D:/NJU_ALL/Sophomore/Digital Logic and Principles of Computer Organization/2021/exp9_board/v
end

assign q=regs[address];

endmodule
```

5. 实验中遇到的问题及解决办法

5.1. 按一下出 11 个字符

加一个计数器，第十一个才输出。

5.2. 输出位置有些许偏移

加一个偏移量调整。

```
ram_out_addr <= (v_addr >> 4) * 71 + ((h_addr+2) / 9);
```

5.3. 用 initial 初始化数组失败

我为什么发现初始化失败，主要是通过七段数码管将值显示在开发板上发现的。

```
integer i;

reg flag2=0;
always @ (posedge PS2_CLK)//写入地址改变
   begin
        if(!flag2)
        begin
        for(i=0;i<31;i=i+1)
          ram_index[i]=4095;//初始值
        flag2=1;

        end
```

只能通过一个标记，只在第一个时钟上升沿赋值了。

5.4. 上下左右方向键会消除掉字符

加一个写入使能端，输入方向键时防止写入。

5.5. 方向键各种写入位置乱跳

解决办法见前文。

6.实验启示和建议

建议给出的 font.txt（或直接给 font.mif）直接可以读入，而不用再写个 c 语言转成可读入的，或者写个脚本语言转成.mif 文件。

希望能给出实验七键盘模块的正确代码，因为就算验收通过了还是有可能错，而键盘输入代码是实验九乃至实验十二最基本的，时序逻辑复杂，出了问题也很难看出来。

这个实验确实收获很多，但是很多时间都在调试综合 debug，浪费了大量时间，感觉这个实验除了上板很难仿真。