数电实验EXP11

刘永鹏 191220070 693901492@qq.com 12.1

1. 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。界面实 现要求可以参考 DOS 字符界面，Window 命令行或 Linux 的字符终端。

基本要求

1. 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。2.
2. 一直按压某个键时，重复输出该字符。
3. 输入至行尾后自动换行，输入回车也换行。

扩展要求（做5个满分）

1. 可以显示光标，建议可以用显示闪烁的竖线或横线作为光标。2.
2. 支持 BackSpace 键删除光标前的字符。3.
3. 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。
4. 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。
5. 支持 Shift 键以及大小写字符输入。
6. 支持方向键移动光标。
7. 在行首显示命令提示符。

2. 实验原理

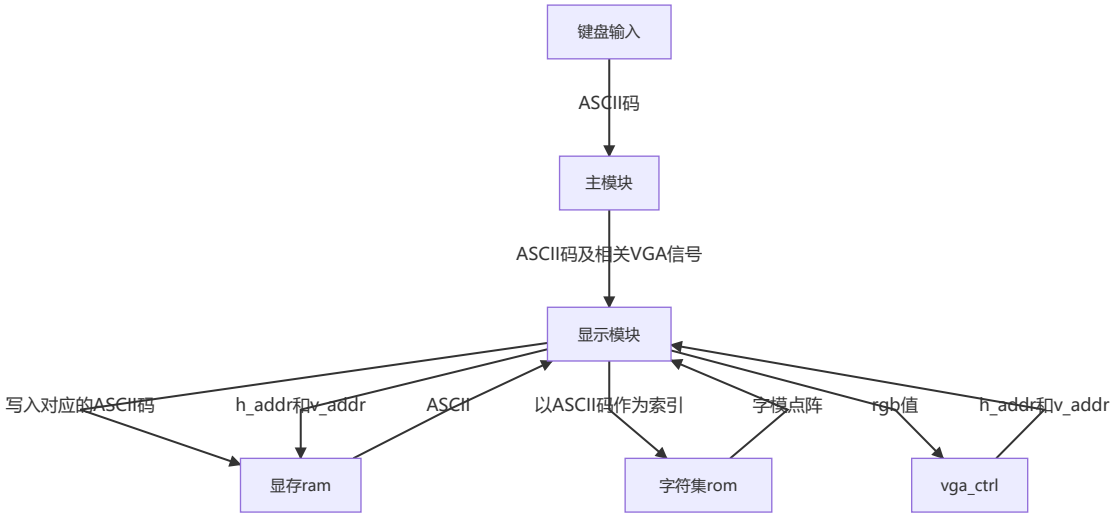
1. 字符显示界面只在屏幕上显示 ASCII 字符，其所需的资源比较少。首先，ASCII 字符用 7bit 表示，共 128 个字符。大部分情况下，我们会用 8bit 来表示 单个字符，所以一般系统会预留 256 个字符。我们可以在系统中预先存储这 256 个字符的字模点阵。
2. 有了字符点阵后，系统就不再需要记录屏幕上每个点的颜色信息了，只需要记录屏幕上显示的 ASCII 字符即可。在显示时，根据当前屏幕位置，确定应 该显示那个字符，再查找对应的字符点阵即可完成显示。
3. 我们的显存此时为30*70，记录每个位置上的ascii码，这减小了整体的内存消耗。

3. 实验环境

systembuilder自动建立项目。

4. 实验代码/实验截图

对于模块划分：

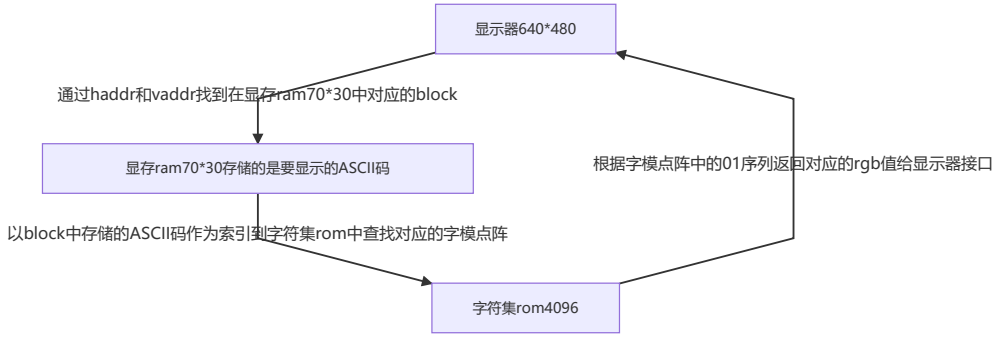


首先是整体上的实验模块划分，划分为三大部分，键盘输入部分，主模块，和显示模块，键盘输入部分基本复用实验八中的键盘模块，将键盘输入得到的ASCII码传递给主模块，而主模块把FPGA上的各种信号以及键盘输入模块产生的ASCII码传递给显示模块，显示模块负责根据改ASCII码，修改相应的显存，实现字符的输入显示效果。

而本实验的关键就在于显示模块的编写，其又分为两大部分，一是静态字符显示，当前的v_addr和h_addr去显存ram中找到应该显示的字符的ASCII码，然后以该ASCII码为索引到字符集rom中读取相应的字模点阵，根据字模点阵的01序列，传送对应的rgb值给vga_ctrl模块，实现对于静态字符的显示，由此可以知道，对输入的显示实际上就是修改显存ram中的数据，转换为对静态字符的显示即可，具体的实现方式在下面具体说明。

静态字符显示部分

静态字符的显示核心在于三个映射，显示器到显存ram的映射，显存ram到字符集rom中的映射，从字符集rom到显示器的映射，正确实现这三个映射就可以在显示器上静态显示字符



而映射的主要代码如下：

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

首先是从显示器的640 * 480像素点映射到显存ram的70 * 30个块,由于每行有70个字符,共30行,因此对于每个vaddr要除以(480/30 = 16)就得到对应的显存ram中的横坐标,再将h_addr/9就得到对应的显存ram中的列坐标,同时由于除法消耗太大,因此除以16就转换为右移4位,提高了计算的效率,确保能在一周期内完成计算,此外由于在ram中使用的是一维地址,因此这里要对横坐标和列坐标做二维到一维的转换,因此其横坐标*70再加上列坐标即得到显存ram中对应的块的地址.

然后根据之前计算的block_addr到显存ram中取出ASCII码(ram_vga_ret),以该ASCII码为索引,到字符集rom中取出对应的字模点阵,但要注意的是,字模点阵是以行存储的,每个字符占16*9大小,因此对于一个ASCII码的显示需要逐行读取字符集rom,因此读取地址address <= (ram_vga_ret << 4) + (v_addr % 16)得到对应字模点阵当前行的9位数据font

最后是从字符集rom到显示器的映射,即字模点阵到rgb值的转换,通过h_addr对9取模得到当前需要显示的像素是从rom中取出的9位数据中的哪一个bit，根据这一个bit是0还是1输出黑色和白色。

由此，实现了对静态字符的显示。

输入字符修改ram

为了实现对输入字符的显示，需要根据键盘输入产生的ASCII码修改显存ram，此时要注意的问题是，vga扫描显存的时钟是很快的，但是控制键盘输入的时钟并没有那么快，因此时序的问题显得尤为重要，在这里我才用了双口ram来设计这个显存，并且用不同的时钟控制它的两个port，这样就可以避免因两个时钟频率不一致而带来的读写时序问题

```
module ram_vga (
    address_a,
    address_b,
    clock_a,
    clock_b,
    data_a,
    data_b,
    wren_a,
    wren_b,
    q_a,
    q_b);

    input   [11:0]  address_a;
    input   [11:0]  address_b;
    input     clock_a;
    input     clock_b;
    input   [7:0]  data_a;
    input   [7:0]  data_b;
    input     wren_a;
    input     wren_b;
    output  [7:0]  q_a;
    output  [7:0]  q_b;
```

```
ram_vga rv(block_addr,index,clk_vga,clk_keyboard,1'b0
,ascii_data,1'b0,1'b1,ram_vga_ret,now_ascii);
```

两个port分别使用clk_vga和clk_keyboard控制，一边用于vga不停访问显存以显示，另一边用于将键盘的输入写入该ram中，而写入的位置由index控制

```
    always @ (posedge clk_keyboard)
    begin
        if(ascii != 0)
        begin
            if(ascii == 8'h0d) index <= index + 70 - (index % 70);
            else index <= index + 1;
        end
        if(index == 2100) index <= 0;
    end
```

特别说明，这是仅仅实现基础功能的版本，后来实现的拓展功能对该部分做了较大修改，这里为了实验报告说明方便特意使用未实现扩展功能的版本来说明

其输入逻辑较为简单，判断输入的ASCII码，如果是回车，index就加70并且减去index%70，这样index就到了下一行的首地址位置，如果是普通非控制字符就正常写入ram并让index+1，由于index加到行尾后再加一实际上就到下一行行首，输入可以正常换行。

至此，基础功能部分已全部实现

拓展功能

由于实现拓展功能的代码是实现了好多个不同的拓展功能的版本,不易拆分出来单独说明,因此该部分的主要代码统一在下图中展示,而相关说明则分别以每个拓展功能来说明,在下面说明每个拓展功能时就不再贴出代码

```
always @(posedge clk_keyboard) begin
    if(ascii != 0)
        begin
            if(cnt == 0 )
            begin
                if(ascii == 8'h0d)begin//hit enter
                      backspace_state <= 0;
                      ram_index[index/70] <= index + 1;
                       index <= index + 70 - (index % 70);
                      enter_state <= 0;
                end
                else if (ascii == 8'h08 ) begin//hit backspace
                        if(backspace_state == 0)begin
                            index <= index;
                            backspace_state <= 1'b1;
                        end
                        else begin
                            if(index % 70 == 0)begin
                                index <= preindex;
                            end
                            else begin
                                index <= index - 1;
                            end
                        end
                      end
                else begin//hit key
                      if(backspace_state == 1 || enter_state == 0)
                        index <= index;
                      else begin
                        if((index + 1) % 70 == 0)begin
                            ram_index[(index-1)/70] <= index;
                        end
```

```
                        index <= index + 1;

                    end
                    backspace_state <= 0;
                    enter_state <= 1;
                end
                cnt <= cnt + 1;
            end
            else if(cnt == 4'd12)begin
                        if(ascii == 8'h0d)
                            index <= index + 70 - (index % 70) - 1;
                        else if (ascii == 8'h08)begin
                            if(index % 70 == 0)begin
                                index <= preindex;
                            end
                            else begin
                                index <= index - 1;
                            end
                        end
                    else begin
                        if((index + 1) % 70 == 0)begin
                            ram_index[(index-1)/70] <= index;
                        end
                            index <= index + 1;
                        end
                    end
            else begin
                index <= index;
                cnt <= cnt + 1;
            end
            ascii_data <= ascii;
        end
    else begin
        index <= index;
          cnt <= 0;
     end

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

更流畅的输入体验

由于该输入的原理为设置一个键盘时钟，每次该上升沿到来时检测是否有输入，如有输入就在键盘上显示，因此，假如我只想输入字符，必须按下按键的时间当且仅当一个周期，这对使用者的体验非常不友好，因此，尽管实验说明文档中并未对此做出要求，但我也对此进行了改进。

实现该功能的关键在于针对按下的第一个键的处理，观察实际使用的键盘发现，在单次按键与长按中实际上是存在一定的延迟的，即第一个按键就算按的时间稍微长了一点也不会出发长按，只有超过一定的时间之后才会触发长按，因此我想到可以用计数器来对输入体验进行改善。

具体而言就是设置一个计数器，每次当clk_keyboard上升沿到来时，如果当前输入有效，则cnt + 1,同时在显示器上显示，然后若该键一直被按下，则在接下来的12个周期内，cnt每周期+1，但是index不改变，即此时仍然只有一个字符，若超过12个周期，则cnt不用继续再加，此时已进入长按状态，index每个周期+1，显示出长按的字符，同时调整时钟频率到正常人类任意按下并松开键盘都可以被检测到，由此，尽管可能每个键按的时间不尽相同，但只要都是在12周期以内，无论按压键盘的快慢，都视为只输入一个字符，这样更接近实际使用的键盘，实现了更流畅的输入体验！

上图中26行到61行的代码与之有关

backspace删除字符

index实际上是指向的最后一个字符的位置,因此当输入的第一个退格时,index不能马上-1,首先要把最后一个字符删除,然后若接收到第二个退格才能减一并删除前一个字符,因此设置了backspace_state,若此时是第一个退格,则index不变,由于已经修改了backspace的ASCII码对应的字模点阵为0,这时向该index位置写入全0，对应的字模点阵为全黑，因此就等价于把最后一个字符删除,之后将backspace_state置为1,之后如果再接收到backspace时,index开始-1，此时的状态实际上是下图这样，即index在最后一个字符之后。

如果是继续删除，这样的状态无任何问题，index继续减一即可，但如果要在删除之后又输入字符，此时就会出现两次输入中间空了一格的情况，因此之前设置的backspace在这又起到了关键作用，如果backspace为1，就表示刚刚经历了一次退格，因此这时又遇到重新输入时index不再增加，在原位置输入，之后把backspace置为0，这样就完全恢复到了没发生删除之前的状态。

删除到行首回到上一行的非空字符位置

该拓展功能的实现关键在于存储每一行非空字符的位置,当删除到行首时,index不再继续-1,而是从存储器中读出上一行的非空字符的位置,并且将index修改为非空字符的位置,而每一行的最后一个非空字符有两个来源,一是正常输入到行尾换行,二是输入回车换行,每次遇到这两种情况时在新开的ram中保存一下当前的index,当删除到行首时从ram中取出上一行的index即可.和该ram有关的代码如下,其余代码见上

```
always @(clk_keyboard)begin
    if(index < 12'd70)begin
        preindex <= 0;
    end
    else begin
        preindex <= ram_index[(index/70)-1];
    end
end
```

其中preindex是记录上一行最后一个非空字符的index

上图中涉及backspace_state和preindex和hit backspace的代码与之有关

shift和大小写字母的显示

由于该功能是在键盘输入部分实现,已在之前的报告中有详尽的说明,由于键盘输入模块会正确处理大小写状态,返回对应的大小写ASCII码,与怎样去显示大写字母关系不大,仅仅是ASCII码的区别.因此该拓展功能自然就已经实现了，与该实验无太大关系.

ASCII字符画的显示

实际上字符画就是通过向显存ram中写入不同的字符来作画,通过实现确定好的mif文件初始化显存,即可实现在显示器上显示字符画.因此唯一的问题就是怎样得到对应的mif文件，经过上网查阅资料，发现其实GitHub上就有开源项目，把对应的字符画转换为mif文件的工程，阅读其相关实现，理解各个接口的含义之后便可把想要生成的字符画转换为mif文件，这样其实就变成了变相的图片显示而已。

3. 进行全编译

5. 实验结果

已完成验收。

6. 问题与解决方案

1. 本次实验有多种不同的时钟,时钟之间的时序问题尤为重要,一不小心就会犯下难以察觉的因为时序产生的错误
2. 对于在同一周期完成的操作，最好都写在同一个always里实现，而不要一个写在时钟驱动的always语句块里，另一个由于较为简单就通过assign直接赋值，这样出现的后果这两个数据不再同步，两个数据不在同一个周期改变，就会出现时序上的错误，带来不易发觉的bug。
3. 该实验工程量较之前的实验较为庞大,一定要在实验前做好模块划分,想清楚每一部分是需要怎样实现,再逐步解决,否则就会陷入迷茫不知道从何下手
4. 遇到问题可以通过观察生活中的实际应用解决,比如这次实验中不知道怎样才能获得我每次输入一个字符,不会因为按压键盘的时间稍微长了一点导致输出两个键这样良好的输入体验,我仔细观察了自己笔记本上的键盘对于这样的输入是什么效果,最终得到了用计数器控制来实现良好输入体验的效果,并最终实现了类似实际键盘输入一样的效果,虽然这不一定是最合理的解决办法,但是是自己想出来的,这个过程本身就很有意义.