实验十一 字符输入界面

实验报告

181860085 汤昊

数字电路与数字系统二班

邮箱：1174639585@qq.com

2019.12.1

一．实验目的

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。

二．实验原理

（1）字模点阵

图中包含 256 个字符，每个字符高 16，宽 9，故用 16 个 9bit 数据存储，一个 bit 是 1 代表相应位置是白色，0 代表相应位置是黑色。实际存储时，因为要保持对齐便于寻址，采用存储 16 个 12bit 数据的方式，其中前三个 bit 都为 0，从低位到高位对应字符从左到右，故共需要 256*16*12=49152bit

(2) 扫描显示

显示器逐行扫描，全部扫描完后回到起点。本实验中，显存存储 ASCII 码，根据行列坐标去显存中读出 ASCII 码，查询子模点阵得到对应字符点阵数据，再根据行列坐标提取出的块内坐标提取出对应位置的数据，决定当前扫描位置的像素值。

(3) 键盘输入

键盘按下时会发送键码，可以根据实验 8 中得到的 times(按键次数)，break(断码)判断是否按下，键码会被翻译成 ASCII 码，这是就可以根据 ASCII 码的类型决定是写入显存还是执行特定操作，如退格，回车等。

三．实验环境及器材

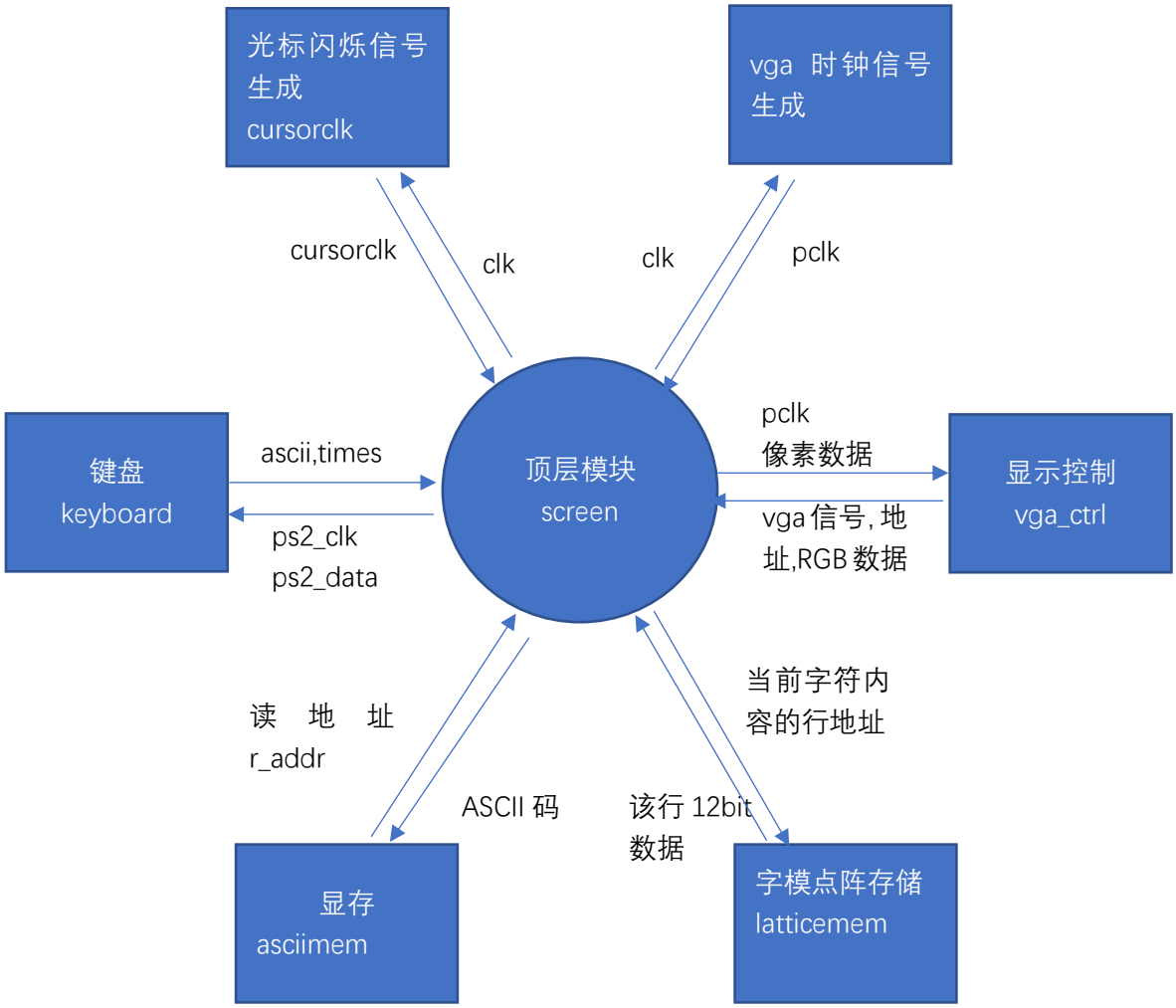
开发软件：quartus prime 17.1

开发器材：DE-standard 开发板

ps2 键盘

vga 接口显示器

四．模块划分



screen：顶层模块，负责接受，处理，传递时钟，vga 和键盘的各种信号以及显示地址转换，完成对显存读写的逻辑

cursorclk：产生频率 1Hz 的时钟，使光标闪烁

clock：产生 25MHz 的 vga 时钟 pclk

key：键盘模块，接受键码并转化为 ASCII 码，对按键计数，而将键码转换为 ASCII 码的操作是由实验 8 中的 display 模块完成的

vga_ctrl：显示器的地址计数，消隐信号产生等，分割输入的像素为 RGB 数据

asciimem：一个大小为 2400*8bit 的 ram，上升沿且写使能时写入，下降沿时读出

latticemem：一个大小为 4096*12 的 rom，在上升沿读出

五．功能实现及仿真模拟

（1）地址转换

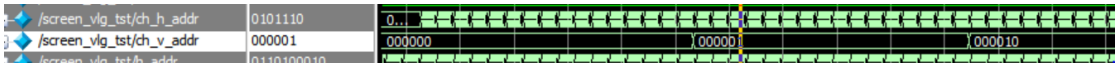
本实验中需用到 6 个地址，分别是字符的行列地址，字符块内的行列地址，显存的读写地址

1）字符的行列地址

640*480 的显示屏可以显示 30 行 70 列字符，为了便于寻址，缩减为 30 行 64 列字符。

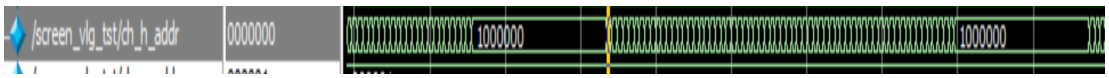
- 字符的行地址：由于 480/30=16，故直接取像素行地址的前 6 位

```
assign ch_v_addr=v_addr[9:4];
```



图中 ch_h_addr 周期性变化 16 次后 ch_v_addr 变化

- 字符的列地址：每个字符宽度为 9，由像素的列地址除 9 即可得到字符列地址，但除法时间代价太大，故转化为计数操作。维护一个 0-8 的循环计数器，在 pclk 的上升沿递增，到 8 时清零，同时列地址 ch_h_addr 加一，到 64 时清零。由于列地址 0 到 63，故以上的操作只在像素列地址在 0-576 时进行



可以看到 ch_h_addr 从 0 一直变化到 64，64 是在消隐时的状态故不影响显示

2）字符块内的行列地址

一个字符块大小为 16*9

- 字符块内的行地址：直接取像素行地址的第 4 位

```
assign in_v_addr=v_addr[3:0];
```

由于是按行扫描，一行的 in_v_addr 都不会变，在 in_h_addr 变化 64*9 次后，in_v_addr 变化

- 字符块内的列地址：得到字符列地址过程中使用的 0-8 循环计数器实际上就对应着块内列地址

in_h_addr 从 0-8 循环变化

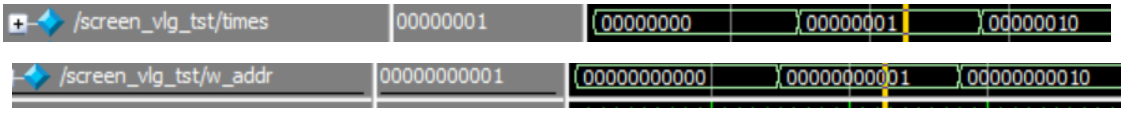
3）显存的读写地址

读地址：由字符的行列地址合成，即字符行地址*64+字符列地址，为了减少时间代价，

乘法用扩充位数操作替代。

```
assign r_addr={ch_v_addr,{6{1'b0}}}+ch_h_addr;
```

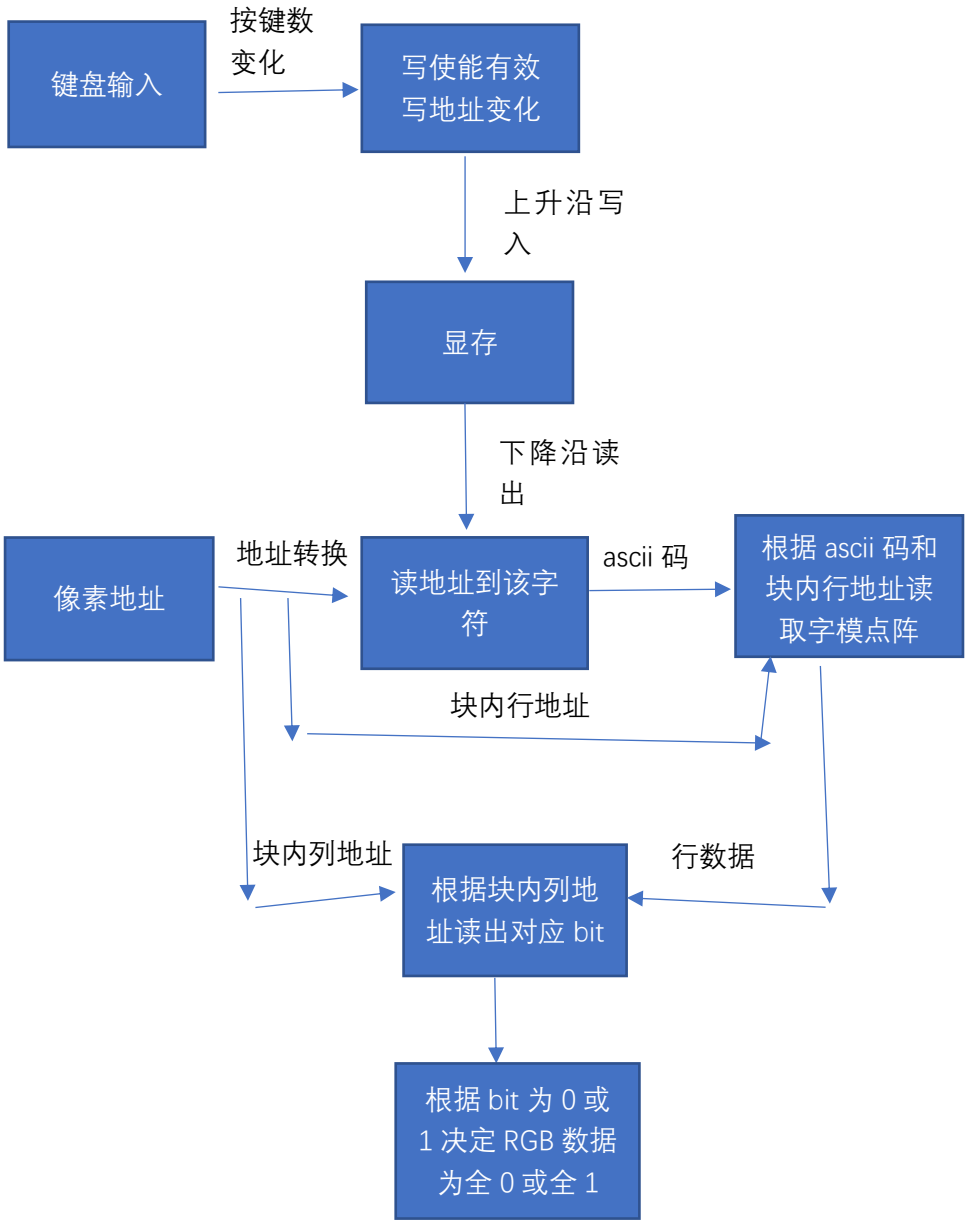
写地址：键盘有输入时写地址要相应变化，为了检测键盘有输入，在顶层模块中保留 times(按键次数)的旧值，每个 clk 检测键盘模块的 times 和旧值是否相同，不同则把 wren 置 1, 相同置 0, 这样在 wren 的上升沿根据解析的 ascii 码决定 w_addr 的变化，如果是退格符，需要减一，如果是换行符需要跳到下一行的首地址，其余地址加一

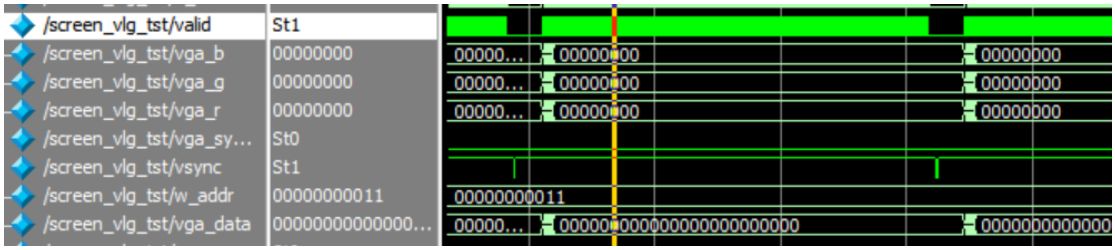


w_addr 与 times 变化同步

（2）基本字符显示

一个字符从输入到显示出的流程如下图所示





仿真中只输入了两个字符，故只在开头有 RGB 数据

（3）大小写及支持 shift 键

之前在键盘模块中已经实现了 shift 键和 caps 键的信号，故只需在键码转换为 ascii 码的过程中加入对 shift 信号和 caps 信号的判断，如果 caps 有效则转换为大写字符 ascii 码，如果 shift 键有效转换 shift 条件的语义，这部分字符包括！@#￥%^&*()等

（4）光标的显示

光标需要在写入地址的下一位显示，且需要以固定频率闪烁。故设置光标区域标志位 cursor，其有效条件是 cursorclk 有效+读地址为写地址的下一位+块内列地址为 0

```
assign cursor=(cursorclk==1)&(r_addr==w_addr+1)&(in_h_addr==0);
```

其中 cursorclk 设置为 1Hz

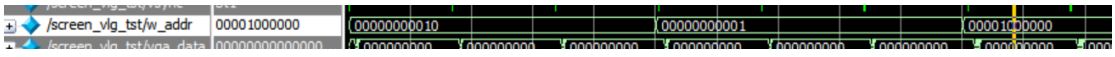
这样在每次由字模点阵的 bit 数据确定 RGB 数据的同时判断 cursor 是否有效，如果 cursor 有效则 RGB 设置为全 1，显示白色。由于 cursorclk 的存在，cursor 的有效频率也是 1Hz，从而产生闪烁效果。

由于 1Hz 太慢，实际测试时改为 1000Hz，可以看到 cursorclk 的和 cursor 的变化

（5）回车换行

由于总列数为 64，因此跳到下一行的第一列只需要将 w_addr 的后 6 位置 0，前 5 位加一即可

```
w_addr[5:0]=6'b000000;
w_addr[10:6]=w_addr[10:6]+1;
```



图中 w_addr 在回车后由 00000000001 变为 0001000000

（6）命令提示符

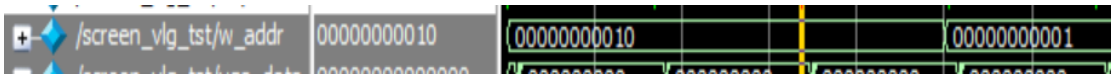
在回车时写地址会跳到下一行的第一列，此时写使能有效，ascii 码值为回车 8'h0D，因此在显存写入时如果 ascii==8'h0D,则替换为写入 8'h24，这样在每次回车时都会写入$命令提示符。第一行的命令提示符由显存的初始化文件完成，即 blank.txt 第一个字符为 8'h24。

```
if(data==8'h0D)
    mem[wraddress]<=8'h24;
```

（7）退格删除功能

如果收到到退格键的 ascii 码，首先 w_addr-1，之后显存在 w_addr+1（此时 w_addr 已减过 1）位置写入 8'h00，这样就删除了原来的字符

```
if(data==8'h08)
    mem[wraddress+1]<=8'h00;
```



图中退格后 w_addr 由 00000000010 减为 00000000001

（8）退格删除回车恢复

顶层模块中维护了一个 entered 数组，用来表示下标行是否使用回车跳到下一行，同时维护了一个 lineend 的数组，用来存储回车前这一行的末尾位置。这样在回车时记下在 entered 数组中记下有回车，在 lineend 数组中记下末尾位置。在执行退格操作时，如果 w_addr 的低六位为全 0，则去 entered 数组中查询上一行是否有回车，如果有 w_addr 变为 lineend 中的值，没有则正常减一

```
entered[w_addr[10:6]]=1'b1;
lineend[w_addr[10:6]]=w_addr;//保存信息
if(w_addr[5:0]==6'd0&&entered[w_addr[10:6]-5'd1]==1)
        w_addr=lineend[w_addr[10:6]-5'd1];//跳到上一行末尾
```

先回车再退格，可以看到 w_addr 恢复到回车前的位置

六．实验反思与总结

- 读地址 r_addr 的转换：
  问题：尝试使用左移 6 位完成 ch_v_addr*64，但 ch_v_addr<<6 的实际结果为 0（ch_v_addr 共六位）
  解决方案：使用位数扩展的操作，即{ch_v_addr,{6{1'b0}}}，将 ch_v_addr 的值扩大 64 倍。

- 字符列地址的计算：
  问题：开始是在每次 h_addr 变化时计数，虽然仿真结果良好，但实际显示结果像素一直在抖动，由于 h_addr 也是由 pclk 触发变化，猜测是时序设计问题
  解决方案：改成在每次 pclk 上升沿计数解决

- 检测键盘输入：
  问题：最初是直接根据 times 的变化，如果 times 变化则判断 ascii 码，进而变化写地址，仿真结果良好但 quartus 在综合时并没有实现这一功能。
  解决方案：后来改为 times 的变化带来 wren 的变化，以 wren 的上升沿作为写地址变化信号解决。

- 字符显示：
  问题：对于显存读写时序设计最初是都在时钟的上升沿，但这样得到的结果会有绿色的边缘，推测是读写的冲突。
  解决方案：改为在下降沿读解决

- 写入显存：
  问题：最初并没有单独的显存模块，而是在顶层模块里加入一个显存寄存器，由于是先 wren 使能，再 w_addr+1 移动，显示结果存在的问题是有时写入会同时写当前地址和前一个地址，即 ssr 在输入 t 后变为 sstt，推测可能因为在同一个模块里，w_addr 还没来得及变化就完成了一次写入。
  解决方案：将显存作为一个模块独立后解决。
- 退格删除时：
  出现了两个问题
  (1) 显示了退格符，这是因为在初始设计时没有阻止向显存中写入退格符的 ASCII 码，解决：在每次写入字符是退格符时写入 0 解决。
  (2) 第一次退格删除了两个字符。当时的设计是不阻止向显存中写入退格符，但在读出时不显示退格符，而删除的原理是 r_addr<=w_addr 的部分才显示，这样在退格时由于写地址回退一格“删除”了一个字符，又由于在回退后的地址写入了退格符，“删除”了一个字符。
  解决：最终改为不写入退格符，删除时向显存写 0 解决

- 回车：
  问题：回车时会在行尾写入命令提示符。
  解决方案：问题和之前写入显存的问题相似，在 w_addr 为变化时就完成了一次写入，显存独立为一个模块后解决。

- 仿真：
  问题：在更改显存大小为 30*64（初始设计为 30*70）后仿真 RGB 数据一直为 0，但实际显示效果良好
  解决方案：经过检查发现原因是 30*70 的显存需要 12 位地址，而 30*64 的显存只需要 11 位地址，问题发生时只更改了顶层模块中的读写地址为 11 位，未更改显存地址的接口，导致仿真时显存的读写地址首位不确定不能正常仿真。这一错位是在仿真的 warning 中显示。统一地址位数后解决