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

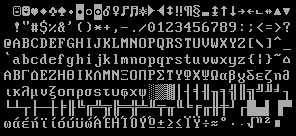
2019.12.1

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

vga时钟信号生成

读地址r\_addr

ASCII码

该行12bit

数据

当前字符内容的行地址

显存

asciimem

字模点阵存储

latticemem

pclk

clk

cursorclk

pclk

像素数据

vga信号，地址,RGB数据

clk

ps2\_clk

ps2\_data

ascii,times

光标闪烁信号生成

cursorclk

显示控制

vga\_ctrl

键盘

keyboard

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

写使能有效

写地址变化

按键数

变化

键盘输入

上升沿写入

显存

下降沿读出

根据ascii码和块内行地址读取字模点阵

ascii码

地址转换

像素地址

读地址到该字符

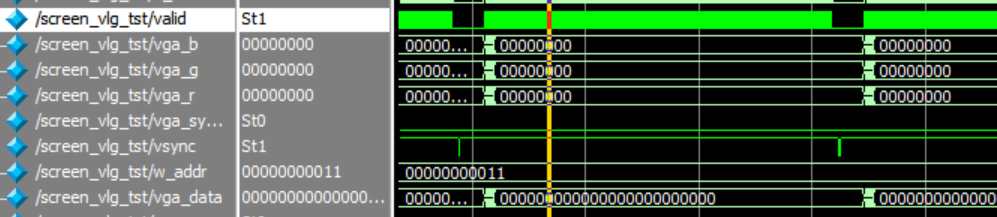
块内列地址

根据bit为0或1决定RGB数据为全0或全1

根据块内列地址读出对应bit

行数据

块内行地址



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

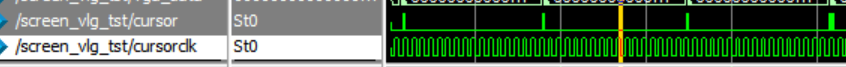
（4）光标的显示

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