南京大学

《数字电路与数字系统实验》实验报告

实验十一： 简单字符交互系统

姓名： 毛彦杰

学号： 191220081

班级：数字电路与数字系统实验 2 班

院系： 计算机科学与技术系

邮箱： 1363818182@qq.com

实验时间： 2020/12/7

一. 实验目的

本实验的主要目的是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。具体要求如下：

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. 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。

可选扩展要求：

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. 支持 Backspace 键删除光标前的字符。
3. Backspace 键删除至本行开始后，再按一次 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。
8. 感兴趣还可以考虑如何实现彩色字符、绘制 ASCII 艺术图或实现类似 Matrix 开头的字符雨效果。

二. 实验原理

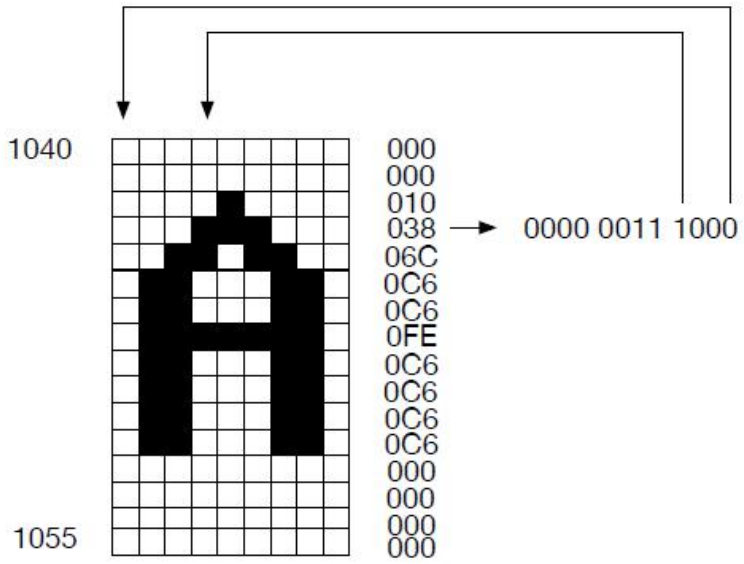
1. 字符显示

从之前的 VGA 图片显示实验，我们体会到图形界面需要大量的资源支持，在 FPGA 上实现高精度、高分辨率的图形界面在资源上有些捉襟见肘。但是，FPGA 上的资源比几十年前的第一代 PC 机要丰富许多。用 FPGA 来实现一个简单的字符输入和显示界面并不难。

字符显示界面旨在屏幕上显示 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，如下图所示

这里每个字符高为 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 256×16×9≈37kbit 的空间即可存储整个点阵。提供的可通过$readmemh 语句读取的点阵文本文件，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)，然后以此类推，最高的 3 个 bit 始终为 0.每个字符 16 行，共 256 个字符。

以字符“A”举例，其 ASCII 的编码是 41h（十进制 65）.因此其字模对应的地址是 16×65=1040(文本文件起始从 1 行开始,因此在第 1041 行）。以 A 字符的第 4 行为例，文本中存储的是 038h，二进制对应是 0000 0011 1000。最低位为 0，所以左边第一个像素为 0，左边第四个到第六个像素为 1.如下图所示（为方便显示颜色黑白颠倒）



有了字符的点阵后，系统就不需要再记录屏幕上每个点的信息了，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。对于 640×480 的屏幕，可以显示 30 行（30×16＝480）， 70 列（70×9 = 630）的 ASCII 字符。系统的显存只需要 30×70 大小，每单元存储 8bit 的 ASCII 字符即可。这样一来，我们的字符显存只需要 2.1k

字节,加上点阵的 6.144k 字节,总共只需要不到 10kByte 的存储,FPGA 片上的存储足够实现了。

2. 系统设计

2.1. 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 ≤ 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，该根据扫描线的行和列的信息，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。

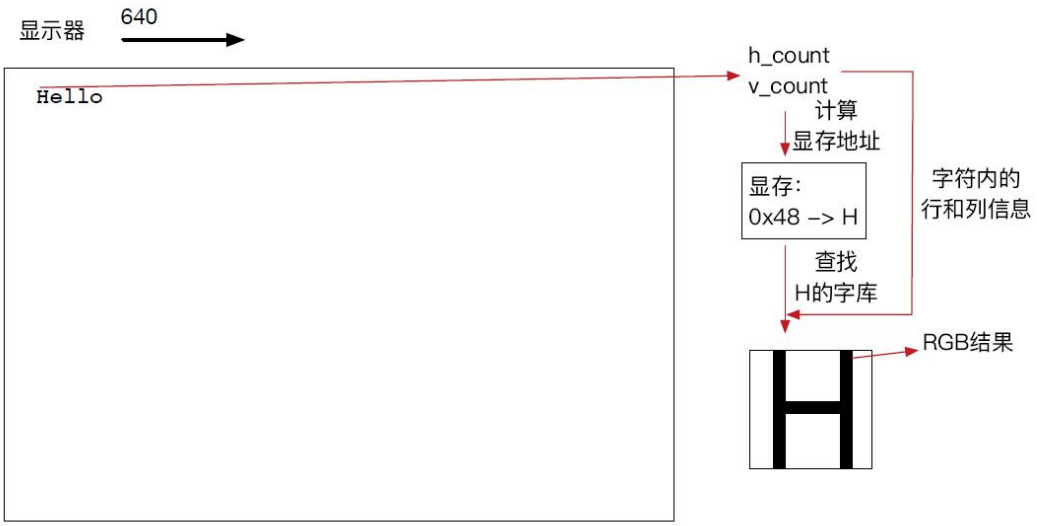
可以将过程总结如下：

a. 根据当前扫描位置，获取对应的字符的 x 和 y 坐标，以及扫描到单个字符点阵内的行列信息

b. 根据字符的 x、y 坐标，查询字符显存，获取对应 ASCII 编码

c. 根据 ASCII 编码和字符内的行信息，查询点阵 ROM，获取对应行的 9bit 数据

d. 根据字符内的列信息，取出对应的 bit，并根据该 bit 设置颜色。此处可以显示黑底白字或者其他彩色字符，只需要按照自己的需求分别设置背景颜色和字符颜色即可。



由于 VGA 的扫描频率是 25MHz，每个点扫过的时间非常短。因此，在扫过一个点的时间内要完成这一系列操作需要仔细地设计时序。首先，FPGA 在这么短的时间内是很难完成乘除法的。因此，设计时要合理选择存储器的大小和寻址方式，将地址计算简化为简单的计数和逻辑操作。其次，对每个点我们要查询多个存储器，每个存储器的查询方式需要合理设计：是采用 RAM 还是 ROM？是上升沿读出还是下降沿读出？这些都有可能影响

系统最终的性能。存在性能问题的显示有可能会出现彩色边缘或者模糊的现象，这些都说明系统实际上存在潜在问题。
如果用寄存器来实现所需的存储器而不是用片上 M10K 或者 MLAB 有可能会占用大量资源，造成 FPGA 资源紧张，编译时间大大增加。

2.2. 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。
存储 ASCII 码的字符下村会经常被 VGA 扫描模块高速读取，而键盘模块需要对显存进行写入，需要注意二者的协调。

三. 实验环境/器材

软件环境：Quartus 17.1 Lite
硬件环境：DE10-Standard 5CSXFC6D6F31C6N、VGA 接口显示器、PS2 接口键盘

四. 实验设计思路

本次实验的成败在于能否将实验八与实验九的代码完美结合。所谓结合并不仅仅只是简单地把代码复制粘贴。这个简单的字符交互系统需要用键盘去写显存，需要 VGA 扫描模块高速读取显存，一定好注意二者时序的配和，并且保证显存相应位置读写的准确性和高效性。

1. 存储器

本次实验显然还需要用到实验七的知识。不管是点阵、显存还是扫描码-ASCII 码转换表，我们都需要用到存储器来存储，并且维护这个存储器来保证各种功能的正确实现。下面将介绍本次实验用到的几个存储器：

1) 用于存储点阵信息的存储器：

```
reg [11:0] dianzhen [4095:0];
```

它存储的内容是课程网站给的 VGA 点阵 RAM 文件。用于 VGA 读取显存得到该点对应的 ASCII 码后用 ASCII 码去查询相应的点阵信息。课程网站的 VGA 点阵 RAM 文件被我保存为“dianzhen.mif”，用它来初始化该存储器。这个存储器不需要进行写操作，我们只用它来读取点阵信息。

2) 用于存储显存的存储器:

```
reg [7:0] xiancun [1919:0];
```

顾名思义，它存储的是显存:)
先解释为什么它有 1920 个存储单元(元素)：显存的每个元素代表一个字符对应的位置。显示器分辨率是 480×630，理应能存下 30×70 个字符(2100 个)，为了避免使用乘除法，我把它变成了 30×64 个字符(1920 个)，当然也可以设计 64 进制的计数器来避免乘除法。它比较重要，也是核心，它不仅要用键盘模块得到 ASCII 码去写，还要用显示器去读某个像素点的 ASCII 码以用于查点阵存储器。

3) 用于存储大小写扫描码-ASCII 码转换表的两兄弟：

```
reg [7:0] ram [255:0];
reg [7:0] rambig [255:0];
```

显然上面那个是弟弟，下面那个是哥哥。在实验八拓展部分已经介绍过他俩了。为了让这份报告更加完美不妨再简要介绍一下：弟弟的是键盘上不需要按 shift 就可以打出来的字符，哥哥存储的是需要按住 shift/用 caps 切换才能打出来的字符。本来考虑只用一个存储，然后想要表示大写英文字母时就 ASCII 码-0x20，但是发现只有英文键适用，其他按键表示的两个字符的 ASCII 码不是这样映射的，于是创建了两个存储器。他们也都是只读的，不用写，只需要用两套 TXT 文件分别初始化，然后在键盘模块从扫描码转换到 ASCII 码的时候读取它们完成转换即可。

4) 用于存储每一行结尾处 x 坐标的存储器：

```
reg [5:0] line [29:0];
```

它有 30 个单元，每个单元存的是显示器每个字符行的最后一个字符的下一个字符位置的横坐标（列坐标）x。为什么要存储这个信息呢？因为我实现了拓展功能中的 Backspace 的功能以及它跨行删除的功能。跨行删除需要退回到最后一个字符后，因此需要记录每个行最后一个字符后的位置，具体实现后面会介绍。

2. VGA 模块和读显存处理

显示器不好好实现，我就是个瞎子，看不到自己有什么 BUG，只有正确地实现了显示器，才能让你看到一些关于写的 BUG、时序的 BUG、逻辑的 BUG 等等。

回顾实验九的知识和代码，我们拿到显示器中 vga_ctrl 模块的 h_addr 和 v_addr 这两个像素点坐标后，需要用它们得到应该显示的该点颜色，用 vga_data 的形式传回给 vga_ctrl 模块，才能让显示器的各个像素点显示出你想要的颜色。具体实现思路如下：

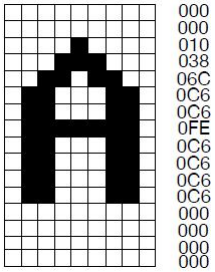
定义 x 为字符横坐标，y 为字符纵坐标；$x_{内}$ 为字符框内横坐标，$y_{内}$ 为字符框内纵坐标

H	E	L	L	O	W	O
R	L	D	!	H	A	!

比如对于这样一个显示器左上角的简化框图，我们对于 A 这个字符（第一行第五列（程序员习惯从 0 计数）），

有 x = 5, y = 1,

而 x 内和 y 内是什么呢？



我们每次得到的 h_addr 和 v_addr 不是字符坐标，而是像素坐标，一个

字符框是由 16 行 9 列的像素点构成的。假设某次给的 h_addr 和 v_addr 需要显示的是上图 A 字符点阵最上面那个“A”的尖尖角，于是，
有 $x_{内} = 4$， $y_{内} = 2$.
接下来我们要考虑如何得到 x，y，x 内,y 内，以及它们有什么用。
因为 y 和 y 内是表示行的坐标，故 y 跟像素行坐标 v_addr 有对应关系，我们可以用 v_addr 计算出 y 和 y 内。又因为一个字符有 16 行像素，故 v_addr/16 即为 y，余数即为 y 内，注意到 16 是 2 的 4 次幂，因此我们可以直接用取对应位数的方式来避免除法和模运算。

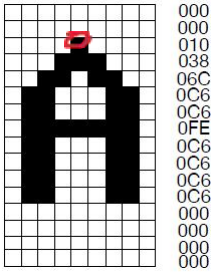
```
y =  v_addr[8:4];
ynei =  v_addr[3:0];
```

对于 x 和 x 内，它们是表示列的坐标，它们跟像素列坐标 h_addr 有对应关系，我们可以用 h_addr 计算出 x 和 x 内。因为一个字符有 9 列，故 h_addr/9 即为 x，余数即为 x 内。这里显然不能“故技重施”，因为 9 不是 2 的整数次幂，他不是我们程序员喜欢的数字。这里我们需要用到年代非常久远的某次实验——计数器，来避免除法和模运算。因为是除以 9 和模 9，那么不妨设置一个 9 进制二位计数器，在每次 vga_clk 上升沿，判断消隐信号无效(vga_blank_n == 1)时计数，因为每次 vga_clk 上升沿且消隐信号 vga_blank_n == 1 时必然会有 h_addr 的自增，于是有以下代码来实现 x 和 x 内的计算：

```
if(VGA_BLANK_N)
begin
   if(xnei == 8)
   begin
      x = x + 1;
      xnei = 0;
   end
   else
      xnei = xnei + 1;
   if(x == 64)
      x = 0;
   else
      x = x;
end
else
begin
   x=0;
   xnei=0;
end
```

得到了 x、y、xnei、ynei，我们需要它们来干嘛呢？
首先，我们需要拿 x 和 y 来读显存。因为显存是按字符来存的，即每个字符框对应这个字符框内字符的 ASCII 码。

H	E	L	L	O	W	O
R	L	D	!	H	A	!



还是这两幅图，比如我们现在的 h_addr 和 v_addr 对应的像素位置是“A”的那个尖尖（红笔涂鸦部分）。

经过计算，x = 5, y = 1, xnei = 4, ynei = 2.

如何读到这个字符框的 ASCII 码是 A 的 0x41 呢？

首先，拿 y×64（为了避免乘法，用的是左移 6 的方式），然后再加上 x 就可以得到他是第几个字符框，每行有 64 个字符，A 是第一行第五个字符，因此加上第零行的 64 个字符，然后加上 5，得到 A 是第 69 个字符框（类似二维数组等效一维数组的下标转换方式）。

```
asc = xiancun[(y << 6) + x];
```

读到 ASCII 后，我们需要得到点阵信息。由于点阵信息的排列是按 ASCII 码大小顺序一个字符接在一个字符下面排成长长的一列长串，而每个字符的长是 16，故我们需要用得到的 ASCII 码×16（避免乘法，用左移 4 位的方式实现），得到点阵存储器对应的这个字符点阵信息开始的左上角的像素位置，然后加上字符框内行偏移量 ynei 找到开始的那一行并读取信息，比如“A”尖尖对应行读取到 0x010 即二进制(0000 0001 0000)，然后用字符框内列偏移量 xnei 定位到像素点，xnei 是 4，所以定位到第 4 位 “1”，因此这个尖尖要用白色显示出来，vga_data 传回 24’hffffff.

```
always @ (posedge vga_clk)
begin

    if(h_addr >= 0 && h_addr < 576)
    begin
       y =  v_addr[8:4];
       ynei =  v_addr[3:0];
       asc = xiancun[(y << 6) + x];
       index = (asc << 4) + ynei;
       dianzhen_line = dianzhen[index];
    end
    else
       vga_data = 24'h000000;

    if(dianzhen_line[xnei])
       vga_data = 24'hffffff;
    else
       vga_data = 24'h000000;

    if(VGA_BLANK_N)
    begin
       if(xnei == 8)
       begin
          x = x + 1;
          xnei = 0;
       end
       else
          xnei = xnei + 1;
       if(x == 64)
          x = 0;
       else
          x = x;
    end
    else
    begin
       x=0;
       xnei=0;
    end
end
endmodule
```

这就是 VGA 与读显存的实现了。

3. 键盘模块与写显存处理

还是实验 8 键盘的老代码，但需要较多的改装。

我们着重考虑的还是这个状态：

```
if((data != 8'b11110000)&&(state == 0))
begin
   //no f0 before this data
   //so this data is eff
```

即这个扫描码不是 F0，它前面的扫描码也不是 F0。那么它显然是一个有效的扫描码，是我按住键盘某个键产生的扫描码，而不是结束的断码。我们需要对它做文章。

首先 countx 和 county 是我定义的两个用于表示写的字符坐标，counth 和 countx 则用于表示写的像素的位置（上述 VGA 模块中的负责读），它们能告诉我我现在写到了哪。

首先，还是用计数器原理来实现：

```
if(counth<576)
begin
   countx = countx + 1;
   counth = counth + 9;
end
else
begin
   line[county] = countx;
   counth = 0;
   countx = 0;
   if(countv < 480)
   begin
      county = county + 1;
      countv = countv + 16;
   end
end
```

在每次需要写显存的时候计数，自然可以记录我写到了哪。
然后我们需要用这个坐标来写显存，当然，写之前要用扫描码取到正确的 ASCII

```
if(capslock^shiftlock == 0)
begin
   ascii = ram[data];
end
else
begin
   ascii = rambig[data];
end
```

这个大小写的判断功能在实验八介绍过了，这里就不重复提及。
然后把得到的 ASCII 码写入对应位置显存：

```
xiancun[(county<<6) + countx] = ascii;
```

然后是回车换行的实现以及拓展功能的部分实现：

```
if(data == 8'h58||data==8'h12||data==8'h59||data==8'h66||data==8'h5a)
begin
   if(data == 8'h66)
   begin
      xiancun[(county<<6) + countx] = 0;
      if(countx > 0)
       countx = countx - 1;
      else
      begin
         county =  county - 1;
         countx = line[county];
      end
   end
   else if(data == 8'h5a)
   begin
      line[county] = countx;
      county = county + 1;
      countx = 0;
   end
end
else
begin
effdata = data;
count = count + 1;
```

分析如下：
首先我们要判断以下是不是读到了左右 Shift、Caps 或者 Backspace，读到它们，不能直接在显存里写它们的 ASCII 码，他们有特定的功能，需

要我们实现。首先介绍一下回车的功能实现：

```
else if(data == 8'h5a)
begin
   line[county] = countx;
   county = county + 1;
   countx = 0;
end
```

这里用到了一个存储器 line，之前提到 line 记录的是每一行最后一个字符结束的位置，因此直接以行坐标为索引，记录此行的 countx 即可，当然，在不使用 enter 换行，而普通打字符打满一行时，也需要维护一下 line，只不过 enter 是让换行提前进行罢了，然后别忘了让行坐标自增，并且让列坐标归零。

接着是判断 Backspace 键，它的功能是删除和退格，因此，我们把目前坐标的显存清零，然后让写坐标倒退一格即可

```
if(data == 8'h66)
begin
   xiancun[(county<<6) + countx] = 0;
   if(countx > 0)
    countx = countx - 1;
   else
   begin
      county =  county - 1;
      countx = line[county];
   end
end
```

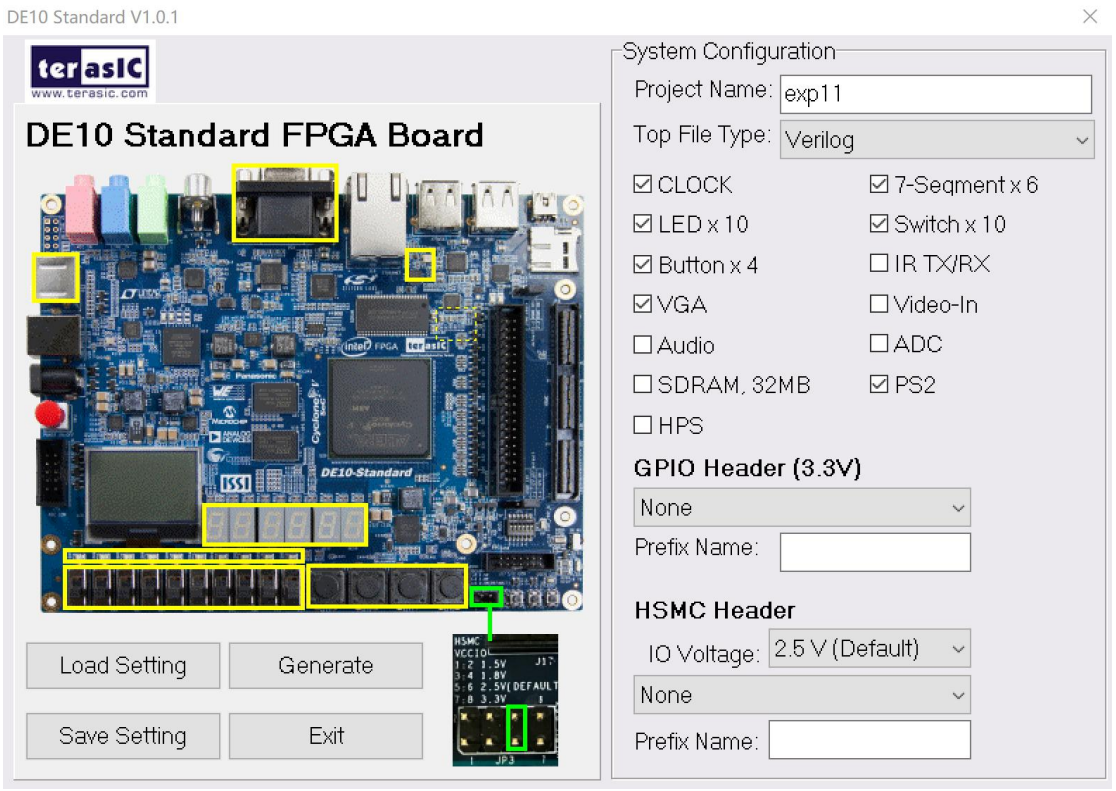
这里需要注意的就是如果删到了前一行怎么办。由于我们已经记录过了前一行的终止字符位置，所以我们让写坐标变成上一行的 line[county] 就行了。

Caps 和 Shift 的转换已经在实验八介绍过了，这里就不多作重复说明了。

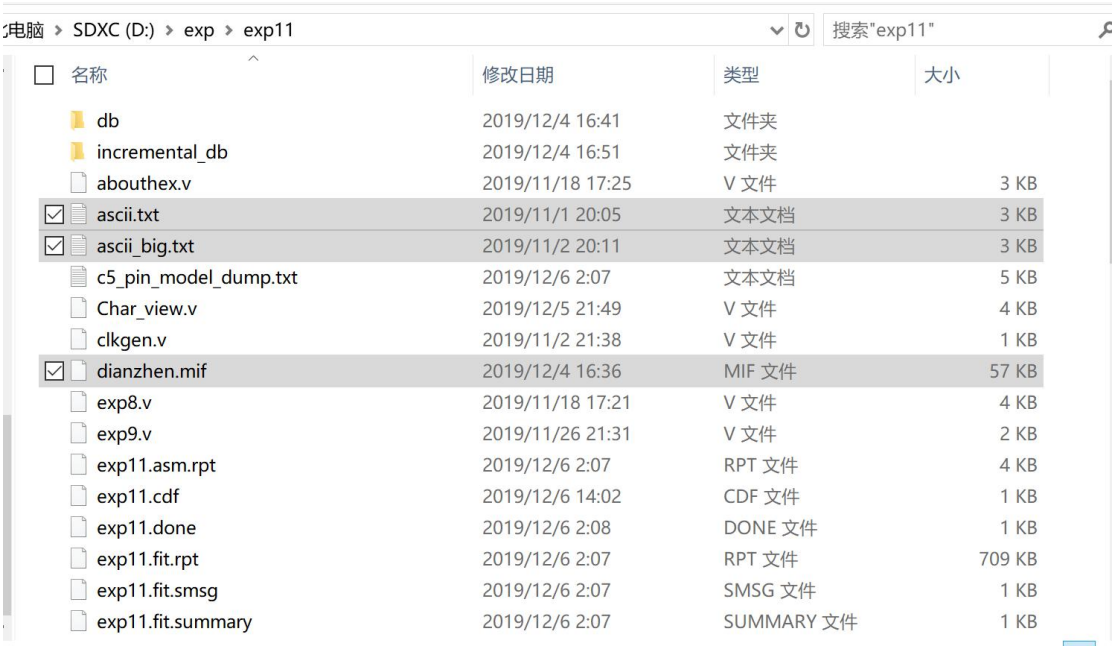
以上就是三个核心部分的实验设计思路叙述。

五. 实验过程

1. 用 System Builder 建立 exp11 的工程文件



2. 保存 VGA 点阵 RAM 文件成.mif 形式，并将各个复用的模块的.v 文件以及.txt 文件保存到正确的路径下。



3. 创建 roms.v 模块，将其按实验设计思路编写。

exp11.v | roms.v* | lock.v

```
module roms(VGA_BLANK_N, clk, ps2_clk, ps2_data, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, LEDR, vga_c
input clk;
input ps2_clk;
input ps2_data;
input vga_clk;
input VGA_BLANK_N;
input [9:0]h_addr;
input [9:0]v_addr;

output reg [23:0] vga_data;
output reg [6:0]HEX5;
output reg [6:0]HEX4;
output reg [6:0]HEX3;
output reg [6:0]HEX2;
output reg [6:0]HEX1;
output reg [6:0]HEX0;
output reg LEDR;

integer x;
integer y;
integer xnei;
integer ynei;
integer index;
integer asc;

wire ovfl;
wire [7:0] data;
wire ready;

(* ram_init_file = "dianzhen.mif" *) reg [11:0] dianzhen [4095:0];//存课程网站给的点阵
reg [7:0] xiancun [1919: 0];//存屏幕上能显示的30*64个字符位置每个位置对应的ASCII码
reg [7:0] ram [255:0];
reg [7:0] rambig [255:0];
reg [5:0] line [29:0];

reg [7:0] count;
reg [7:0] effdata;
reg nextdata_n;
reg state;
reg capslock;
reg shiftlock;
reg [9:0] counth;
reg [9:0] countv;
reg [6:0] countx;
```

exp11.v | roms.v* | lock.v

```
reg [9:0] countv;
reg [6:0] countx;
reg [10:0] county;
reg [7:0] ascii;
reg [11:0] dianzhen_line;

initial
begin
vga_data = 0;
x = 0;
y = 0;
xnei = 0;
ynei = 0;
index = 0;
asc = 0;
counth = 0;
countv = 0;
countx = 0;
county = 0;
ascii = 0;
dianzhen_line = 0;
effdata = 0;
capslock = 0;
shiftlock = 0;
state = 0;
count = 0;
counth = 0;
countv = 0;
end

initial
begin
    $readmemh("D:/exp/exp11/ascii.txt", ram, 0, 255);
    $readmemh("D:/exp/exp11/ascii_big.txt", rambig, 0, 255);
end

always @ (posedge clk)
begin
    if(ready==1)
    begin
    //todo
        if(data == 8'h58)
        begin
            //caps lock
```

exp11.v | roms.v* | lock.v

```
85      begin
86         //caps lock
87         if(state == 0 && capslock == 0)
88         begin
89            //lock is eff, from 0 to 1
90            capslock = 1;
91         end
92         else if(state == 1 && capslock == 0)
93         begin
94            //pre data is f0, this data is not eff
95            capslock = 0;
96         end
97         else if(state == 1 && capslock == 1)
98         begin
99            capslock = 1;
100        end
101        else if(state == 0 && capslock == 1)
102        begin
103           //lock is eff, from 1 to 0
104           capslock = 0;
105        end
106     end
107     if(data == 8'h12 || data == 8'h59)
108     begin
109        //shift lock
110        if(state == 0 && shiftlock == 0)
111        begin
112           //lock is eff, from 0 to 1
113           shiftlock = 1;
114        end
115        else if(state == 1 && shiftlock == 0)
116        begin
117           //pre data is f0, this data is not eff
118           shiftlock = 1;
119        end
120        else if(state == 1 && shiftlock == 1)
121        begin
122           shiftlock = 0;
123        end
124        else if(state == 0 && shiftlock == 1)
125        begin
126           //lock is eff, from 1 to 0
127           shiftlock = 0;
128        end
```

exp11.v | roms.v* | lock.v

```
127           shiftlock = 0;
128        end
129     end
130     if((data != 8'b11110000)&&(state == 0))
131     begin
132        //no f0 before this data
133        //so this data is eff
134        if(data == 8'h58||data==8'h12||data==8'h59||data==8'h66||data==8'h5a)
135        begin
136           if(data == 8'h66)
137           begin
138              xiancun[(county<<6) + countx] = 0;
139              if(countx > 0)
140               countx = countx - 1;
141              else
142              begin
143                 county =  county - 1;
144                 countx = line[county];
145              end
146           end
147           else if(data == 8'h5a)
148           begin
149              line[county] = countx;
150              county = county + 1;
151              countx = 0;
152           end
153        end
154        else
155        begin
156        effdata = data;
157        count = count + 1;
158        if(counth<576)
159        begin
160           countx = countx + 1;
161           counth = counth + 9;
162        end
163        else
164        begin
165           line[county] = countx;
166           counth = 0;
167           countx = 0;
168           if(countv < 480)
169           begin
170              countv = countv + 1;
```

exp11.v | roms.v* | lock.v

```
169             begin
170                 county = county + 1;
171                 countv = countv + 16;
172             end
173         end
174         if(capslock^shiftlock == 0)
175         begin
176             ascii = ram[data];
177         end
178         else
179         begin
180             ascii = rambig[data];
181         end
182         xiancun[(county<<6) + countx] = ascii;
183         end
184     end
185     else if((data == 8'b11110000)&&(state == 0))
186     begin
187         //this data is f0
188         //so this data is eff
189         //and let the count ++
190         //and let the state become 1 to show this f0
191         effdata = data;
192         //count = count + 1;
193         state = 1;
194     end
195     else if((data != 8'b11110000)&&(state == 1))
196     begin
197         //this data is not f0 and predata is f0
198         //so this data is not eff
199         //and let the state become 0 to show this is not f0
200         state = 0;
201         effdata = 8'b11110000;
202     end
203   //done
204     nextdata_n=0;
205     end
206   end
207
208
209 ps2_keyboard pk(clk, 1, ps2_clk, ps2_data, data, ready, nextdata_n, ovfl);
210 abouthex ah(state, count, effdata, ascii, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
211 lock lk(shiftlock, capslock, LEDR);
212
```

exp11.v | roms.v* | lock.v

```
211 lock lk(shiftlock, capslock, LEDR);
212
213 always @ (posedge vga_clk)
214 begin
215
216   if(h_addr >= 0 && h_addr < 576)
217   begin
218     y =  v_addr[8:4];
219     ynei =  v_addr[3:0];
220     asc = xiancun[(y << 6) + x];
221     index = (asc << 4) + ynei;
222     dianzhen_line = dianzhen[index];
223   end
224   else
225     vga_data = 24'h000000;
226
227   if(dianzhen_line[xnei])
228     vga_data = 24'hffffff;
229   else
230     vga_data = 24'h000000;
231
232   if(VGA_BLANK_N)
233   begin
234     if(xnei == 8)
235     begin
236       x = x + 1;
237       xnei = 0;
238     end
239     else
240       xnei = xnei + 1;
241     if(x == 64)
242       x = 0;
243     else
244       x = x;
245   end
246   else
247   begin
248     x=0;
249     xnei=0;
250   end
251 end
252 endmodule
253
```

4. 修改原工程顶层文件，调用 roms.v 模块、clkgen.v 模块以及 vga_ctrl.v

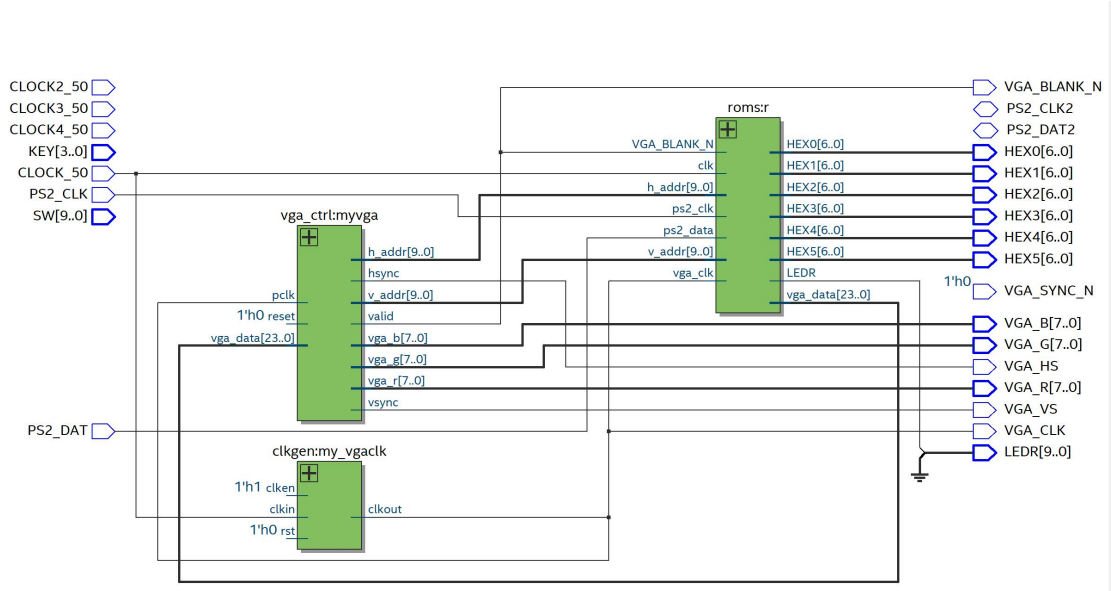
模块，并把 VGA_SYNC_N 置 0。

exp11.v | roms.v* | lock.v

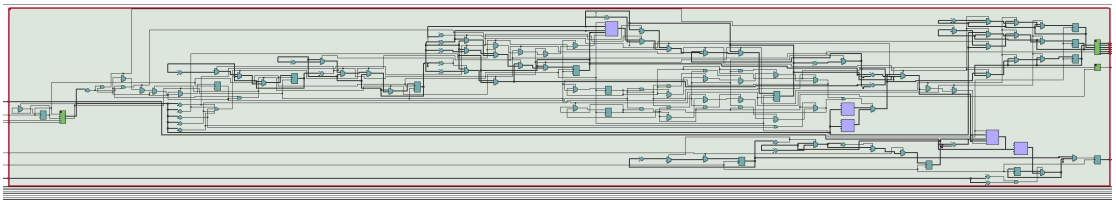
```
46  );
47
48
49
50  //=======================================================
51  //  REG/WIRE declarations
52  //=======================================================
53  wire [23:0] vga_data;
54  wire [9:0] h_addr;
55  wire [9:0] v_addr;
56  //module use_kb(clk, ps2_clk, ps2_data);
57  //use_kb ukb(CLOCK_50, PS2_CLK, PS2_DAT, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, LEDR[9]);
58  //roms(clk, ps2_clk, ps2_data, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, LEDR, vga_clk, VGA_HS, VGA_VS,
59 roms r(VGA_BLANK_N, CLOCK_50, PS2_CLK, PS2_DAT, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0,
60  LEDR[9], VGA_CLK, vga_data,h_addr, v_addr);
61
62  clkgen #(25000000) my_vgaclk  (CLOCK_50, 1'b0, 1'b1, VGA_CLK);
63
64 vga_ctrl myvga(
65  VGA_CLK,
66  1'b0,
67  vga_data,
68  h_addr,
69  v_addr,
70  VGA_HS,
71  VGA_VS,
72  VGA_BLANK_N,
73  VGA_R,
74  VGA_G,
75  VGA_B
76  );
77
78  //module use_vga(clk, VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_R, VGA_G, VGA_B);
79  //use_vga uvga(CLOCK_50, VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_N, VGA_R, VGA_G, VGA_B);
80
81  //=======================================================
82  //  Structural coding
83  //=======================================================
84
85  assign VGA_SYNC_N = 0;
86
87  endmodule
88
```

5. 编译并通过，烧上开发板，用键盘和显示器进行测试，然后遇到 bug，跳回第 3 步（大概经历了几十次吧:)）

附：RTC Viewer 给出的逻辑电路图



本次实验实现的 roms.v 对应的部分：



六. 实验中遇到的问题和解决方法

1. 显示器一直是黑屏没有回显：
   原因就是没有处理好参数是 input 还是 output，仔细检查以后发现很多 vga 相关的东西我都当作 output 处理了，甚至 vga_clk 都是个 output，这显然是不正确的。
2. 时序问题：
   最开始把 ASCII 码的获取沿用之前实验八的模块封装起来，导致用 ASCII 写显存的时候不能保证这个 ASCII 是新读到的，于是果断放弃了这个模块，直接用语句获取 ASCII，然后在执行写显存，这样就保证了写显存是用的 ASCII 是新出炉的 ASCII 了。

七. 启示和感悟

最后一个小实验终于结束了，但我觉得实现这个小实验真的挺不容易的！感觉收货满满！
对于实验手册的这一部分：

> 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。

一开始还觉得到时候写这里会很麻烦，质疑它为什么左右顺序不符合常理，结果发现其实这是高明的做法，牛！

八、意见与建议

实验手册中的实例对学习 verilog 语句使用与 quartus 相关操作有巨大帮助,希望能在学习新知识的同时接触更多相关例子来加深对新知的理解与应用能力。

希望自己能加强对网上知识的搜索自学能力，这样可以迅速掌握新的知识。

非常感谢老师和主教哥哥们的指点！！谢谢!