1、实验目的

理解键盘输出字符的扫描原理。

理解多个模块之间的交互和接口的设计。

2、实验原理

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 25 个字符的字模点阵，如下图所示：

上面图片每个字符高为 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。已经提供了可通过$readmemh 语句读取的点阵文本文件 my_font.txt，其中每 3 个 16 进制数表示单个字符的一行，该行的 9 个点中的最左边点在 16 进制数的最低位，然后依次类推，最高三位始终为 0。每个字符 16 行，共 256 个字符。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，确定应该显示哪个字符，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，字符显存只需要 2.1kByte，加上点阵的 5-6kByte，总共只需要不到 10kByte 的存储，FPGA 片上的存储足够实现。

3、实验环境 / 器材

环境：Quartus Prime 17.1；器材：DE10-Standard FPGA 开发板，VGA 接线，VGA 接口显示器，ps2 键盘。

4、流程图及程序代码

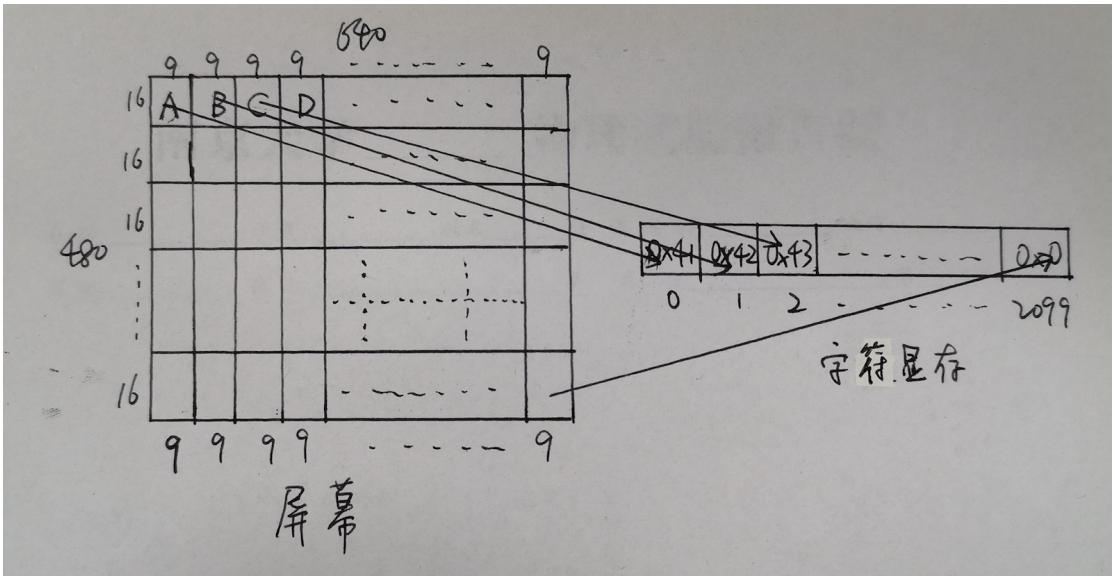
首先设计整体的逻辑框架。最终要实现的效果是要按下键盘上的键，然后在屏幕上输出对应的字符。因此大框架内需要三个核心模块：处理键盘逻辑的键盘模块，处理 VGA 显示器逻辑的 VGA 模块以及实现 VGA 和键盘之间交互连接的顶层实体模块。键盘模块的主要功能就是处理按下键盘发送的键码，之后向外面传出转换好的 ascii 码以及各种标志位信息。VGA 模块主要作用是颜色控制器，实现上层模块控制的颜色输入，输出对应的红绿蓝三色的 8 位二进制信号以及芯片所需的各种标志位信息，传送到开发板上的数模转换器，转换成模拟信号，经 VGA 接口送入显示器中，显示相应控制下的内容。顶层实体模块主要是处理 VGA 模块送出的扫描点行列信息，根据行列信息以及键盘的 ascii 码查询对应字模点阵文件，得到对应行列的像素点是黑色还是白色，配置 VGA 接口需要使用的颜色编码。另外还有一些额外为了实现上述功能需要创建的模块，如由开发板 50MHz 时钟转数模转换器需要的 25MHz 时钟的模块，由键码转化为对应 ascii 码的模块以及存储要打印到屏幕上的字模的显存 IP 核 RAM 模块。

由于在实现上按照上述设计会出现一些问题，因此将其中的一些功能在模块之间交换、修改，形成下面真正实验时上述三个主要模块实现的设计：

首先设计 VGA 模块（该模块名称为 newvga_ctrl）。VGA 模块继承实验九中的 vga_ctrl，该模块可以处理由外部发送的特定像素点的颜色信息，转化为要输出到芯片上的各种信息，并且向外部传出当前扫描到的行列信息以便外部根据行列信息设定颜色信息。此模块需要的外部输入输出是：pclk 为分频器转换好的 25MHz 的 VGA 时钟输入（分频器后面会讲），reset 为重置信号，高电平有效，vga_data 为上层模块用来传入要在显示器上相应像素点显示颜色信号的 24 位编码，包含了红绿蓝各 8 位编码。输出 h_addr，v_addr 用于上层模块，是当前有效像素坐标的指示；hysnc，vysnc 输出作为同步信号到开发板上 VGA DAC ADV7123 芯片，对应芯片的 VGA_HS 信号和 VGA_VS 信号；valid 输出作为消隐信号对应芯片上的 VGA_BLANK_N；vga_r、vga_g、vga_b 输出作为三种颜色的信号，对应到芯片上的三种颜色输入引脚。上面这些是实验九中 VGA 模块的输入输出，这里在原来的基础上添加以下输入输出：clk_50，50MHz 时钟输入，用于字符显存的 IP 核时钟激励；clk_25，25MHz 时钟输入，用于字符显存的 IP 核时钟激励；en，使能端输入，用于 IP 核写使能；waddr，写地址输入，用于 IP 核；raddr，读地址输入，用于 IP 核；asciicode，键盘处理好的 ascii 码输入，用于 IP 核；vgaasciicode，扫描到的对应字符显存的 ascii 码输出：

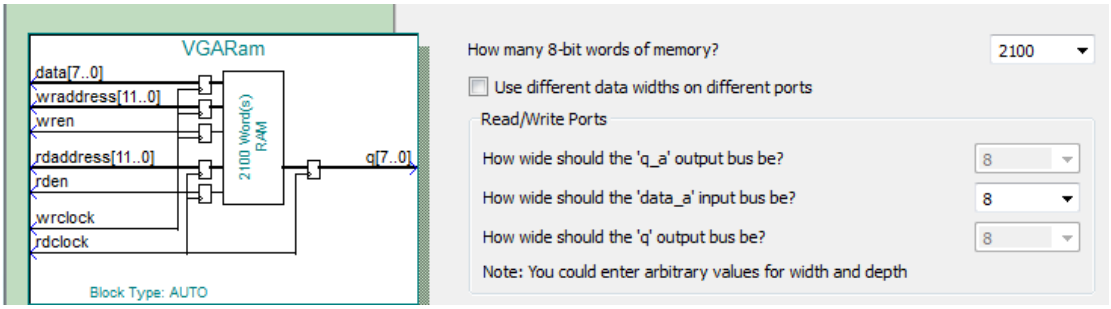
```
module newvga_ctrl(
input pclk,
input reset,
input clk50,
input clk25,
input en,
input [11:0] waddr,
input [11:0] raddr,
input [7:0] asciicode,
input [23:0] vga_data,
output [9:0] h_addr,
output [9:0] v_addr,
output hsync,
output vsync,
output valid,
output [7:0] vga_r,
output [7:0] vga_g,
output [7:0] vga_b,
output [7:0] vgaasciicode
);
parameter h_frontporch = 96;
parameter h_active = 144;
parameter h_backporch = 784;
parameter h_total = 800;
parameter v_frontporch = 2;
parameter v_active = 35;
parameter v_backporch = 515;
parameter v_total = 525;
reg [9:0] x_cnt;
reg [9:0] y_cnt;

wire h_valid;
wire v_valid;
```

剩下的代码在原来 vga_ctrl 的基础上增加了一个 IP 核的模块。这个模块就是字符显存。由于屏幕是 640*480 的，而每个字模想打印出来需要 9*16 的大小，因此在屏幕上字模最多打印出 30 行，70 列，每一个字模都有一个唯一的 ascii 码与之对应。因此为了知道在屏幕上特定的行列要打印哪个字模，需要创建这样一个 30*70 的字符显存，其中每一个元素都是一个 8 位 ascii 码，指示屏幕上对应的位置的 ascii 码信息。也就是说，我们将 480 行，640 列的屏幕划分为 30 行，70 列的数组（630 列～640 列没有实际作用），这样的话原来 480 行，640 列中每 16 行以及每 9 列构成一个数组的元素。每个元素其实就是一个字符，有一个 ascii 码与之对应。因此可以创建一个这样的每个元素为 8bits 的有 2100 个元素的数组来指示屏幕上需要打印的字符信息。屏幕与字符显存映射关系如下图：

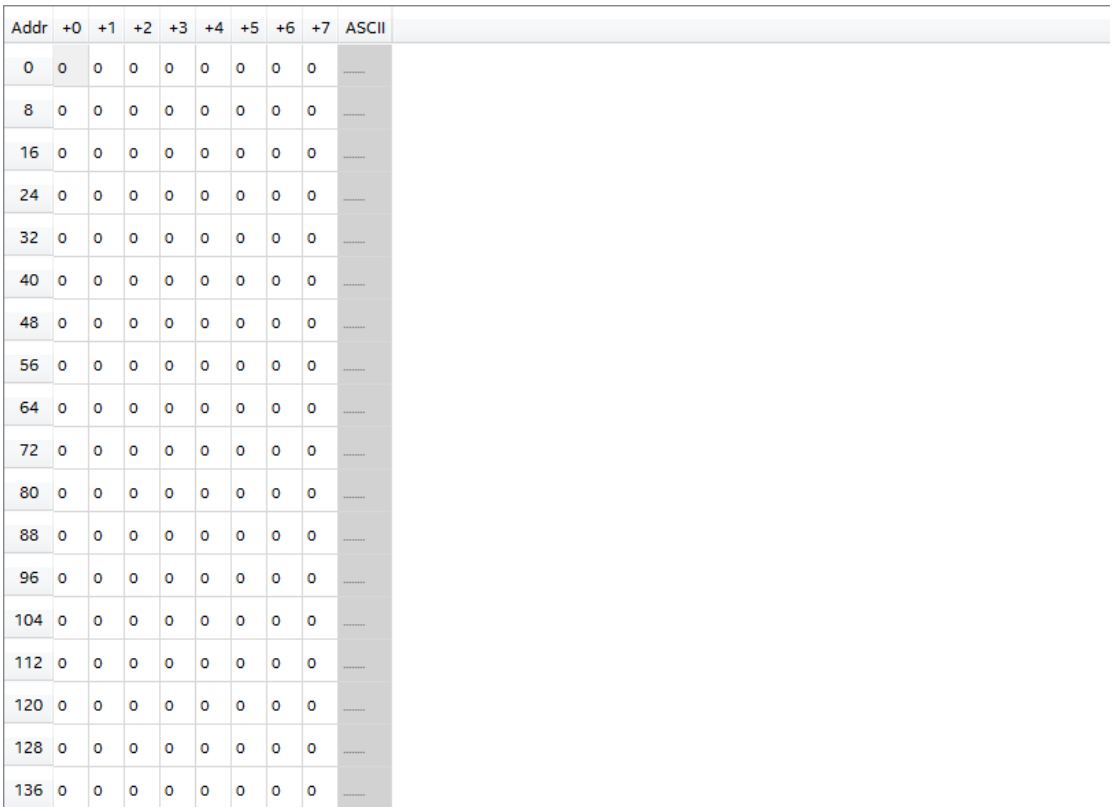


可以看到屏幕上特定的一块区域与字符显存特定元素一一对应。

那么为什么这里实现的时候利用了 IP 核生成的 RAM 而不是直接用数组呢？原因在于如果用数组（寄存器类型），会占用大量资源，造成 FPGA 资源紧张，编译时间大大增加。这也与字符显存需要频繁的写入有关。一开始本来用的是寄存器类型的数组，编译时间都在 10 分钟以上，并且会出现一些复杂的时序逻辑错误，很不好调试。后来改为 IP 核的 RAM，编译时间缩短为 1 分钟，且避免了一些时序逻辑错误。利用 IP 核生成 RAM 实现字符显存。命名为 VGARam。写数据以及读数据都是 8 位 ascii 码，一共有 2100 个元素，设置读写使能，读时钟和写时钟不同：



利用下面的 mif 文件（展示部分）初始化 IP 核，使得一开始开机的时候全屏黑色，也使得已经输入到屏幕上的字符之后的地方时刻保持黑色：



Addr	+0	+1	+2	+3	+4	+5	+6	+7	ASCII
0	0	0	0	0	0	0	0	0	.....
8	0	0	0	0	0	0	0	0	.....
16	0	0	0	0	0	0	0	0	.....
24	0	0	0	0	0	0	0	0	.....
32	0	0	0	0	0	0	0	0	.....
40	0	0	0	0	0	0	0	0	.....
48	0	0	0	0	0	0	0	0	.....
56	0	0	0	0	0	0	0	0	.....
64	0	0	0	0	0	0	0	0	.....
72	0	0	0	0	0	0	0	0	.....
80	0	0	0	0	0	0	0	0	.....
88	0	0	0	0	0	0	0	0	.....
96	0	0	0	0	0	0	0	0	.....
104	0	0	0	0	0	0	0	0	.....
112	0	0	0	0	0	0	0	0	.....
120	0	0	0	0	0	0	0	0	.....
128	0	0	0	0	0	0	0	0	.....
136	0	0	0	0	0	0	0	0	.....

这样就得到了字符显存的 RAM（自动生成的部分代码）：

```
module VGARam (
    data,
    rdaddress,
    rdclock,
    rden,
    wraddress,
    wrclock,
    wren,
    q);

    input [7:0]  data;
    input [11:0]  rdaddress;
    input   rdclock;
    input   rden;
    input [11:0]  wraddress;
    input   wrclock;
    input   wren;
    output   [7:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1    rden;
    tri1    wrclock;
    tri0    wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [7:0] sub_wire0;
    wire [7:0] q = sub_wire0[7:0];
```

其中 rdaddress 为读数据地址，rdclock 为读数据的激励时钟，rden 为读使能，q 为读出的数据；data 为要写入的数据，wraddress 为写数据地址，wrclock 为写数据的激励时钟，wren 为写使能。

VGA 模块还需要一个 25MHz 的时钟输入，这个时钟可以通过开发板 50MHz 时钟转化而来。设定模块 clkgen，产生 25MHz 时钟，代码与实验九一样：

```
module clkgen(
input clkin,//input clock
input rst,//reset
input clken,//enable
output reg clkout
);
parameter clk_freq=1000;
parameter countlimit=50000000/2/clk_freq;
reg[31:0] clkcount;
always @ (posedge clkin)
   if(rst)
   begin
      clkcount=0;
      clkout=1'b0;
   end
   else
   begin
      if(clken)
      begin
         clkcount=clkcount+1;
         if(clkcount>=countlimit)
         begin
            clkcount=32'd0;
            clkout=~clkout;
         end
         else
            clkout=clkout;
      end
      else
      begin
         clkcount=clkcount;
         clkout=clkout;
      end
   end
endmodule
```

rst 为清零端，高位有效，clkout 即为 25MHz 时钟。

VGA 模块中主要的代码部分与实验九基本相同，添加了一个对于字符显存模块的调用。即将外界传入的对应的写入数据 asciicode，读地址 raddr，读时钟 clk_25，读使能常置为 1（表示一直读），写地址 waddr，写时钟 clk_50，写使能

en，通过调用 RAM 得到读出的数据 vgaasciicode，作为 VGA 模块最核心的输出传给顶层实体文件。这里的 vgaasciicode 相当于上面映射关系图中字符显存中的 ascii 码，对于每一个屏幕行列坐标，都可以找到一个对应的 ascii 码值，即 vgaasciicode。注意这里写时钟比读时钟快了一倍，这样可以保证写入的数据可以相对读数据更快的写入，相当于更快的刷新，避免时序上的错误。下面是 VGA 模块剩余的代码部分：

```
always @(posedge reset or posedge pclk)
if (reset == 1'b1)
    x_cnt <= 1;
else
begin
    if (x_cnt == h_total)
        x_cnt <= 1;
    else
        x_cnt <= x_cnt + 10'd1;
end
always @(posedge pclk)
if (reset == 1'b1)
    y_cnt <= 1;
else
begin
    if (y_cnt == v_total & x_cnt == h_total)
        y_cnt <= 1;
    else if (x_cnt == h_total)
        y_cnt <= y_cnt + 10'd1;
end

VGARam vgaram(
    .data(asciicode),
    .rdaddress(raddr),
    .rdclock(clk25),
    .rden(1'b1),
    .wraddress(waddr),
    .wrclock(clk50),
    .wren(en),
    .q(vgaasciicode)
);

assign hsync = (x_cnt > h_frontporch);
assign vsync = (y_cnt > v_frontporch);
assign h_valid = (x_cnt > h_active) & (x_cnt <= h_backporch);
assign v_valid = (y_cnt > v_active) & (y_cnt <= v_backporch);
assign valid = h_valid & v_valid;
assign h_addr = h_valid ? (x_cnt - 10'd144) : {10{1'b0}};
assign v_addr = v_valid ? (y_cnt - 10'd35) : {10{1'b0}};
assign vga_r = vga_data[23:16];
assign vga_g = vga_data[15:8];
assign vga_b = vga_data[7:0];
endmodule
```

然后是键盘模块（命名为 newps2board）。键盘模块继承了实验八中的键盘处理模块，并以其为基础做了代码扩展。输入数据如下：clk 为开发板系统 50Hz 时钟，clrn 为清零端（高电平有效），ps2_clk 是键盘发送的时钟信号，ps2_data 是键盘在其时钟信号下降沿发送的数据信息。中间变量如下：buffer 是接收到的连续 10 位键盘数据，其中包括开始的 0，中间的 8 位键码信息以及奇偶校验位（最后还有一个 1，被抛掉），作为缓冲信息，count 是一个计数器，负责计算连续从键盘接收到多少位数据（12 位算一组数据）。添加的中间变量后面会一个一个提到。输出有 enOut，作为字符显存写使能，codeOut 是当前输入的有效键码转化为的 ascii 码；ascii 为调试用输出，这里忽略；waddrOut 用作字符显存的写地址，ptrOut 指示当前即将要写的位置，后续实现光标功能时使用：

```
module newkeyboard(clk,clrn,ps2_clk,ps2_data,codeOut,enOut,waddrOut,ascii,ptrOut);
input clk,clrn,ps2_clk,ps2_data;
reg [9:0] buffer;
reg [3:0] count;
reg [2:0] ps2_clk_sync;
reg isEnd;
reg isShift;
reg delEnter;
reg isE0;
reg [7:0] data;
reg [11:0] ptr;
reg [11:0] waddr;
reg [11:0] tempwaddr;
wire [7:0] waddr1;
wire [11:0] waddr2;
wire [11:0] waddr3;
wire [7:0] ptr1;
wire [11:0] ptr2;
wire [4:0] ptr3;
wire [11:0] ptr4;
wire [11:0] ptr5;
wire [7:0] asciicode;
reg en;
reg [7:0] back [4:0];
output enOut;
output [7:0] codeOut;
output [7:0] ascii;
output [11:0] waddrOut;
output [11:0] ptrOut;
```

首先设计键盘模块与字符显存的关系。可以确定，字符显存的数据写入完全取决于键盘输入，而屏幕显示的字模又取决于字符显存。因此可以得到下面的逻辑：敲击键盘，键盘发送键码，键盘处理模块判断键码有效之后，发出相关信息给 VGA 模块，VGA 模块写对应的字符显存，屏幕上不断刷新显示相应的结果。屏幕上看到的输入一个字符之后屏幕上会多一个字读原因是字符显存中有一个元素变成了字符的 ascii 码，而这里写入的 ascii 码就是刚刚键盘输入的有效键码转化来的。因此，当键盘给出一个键码，并且该键码属于有效输入（不是 F0，回车，删除键等）的字符时，就需要将一个指向字符显存中该写入的位置（即写地址）传出（对应 waddr），并且将写使能 en 置 1（有效）传出，并且将键码转化为 ascii 码后将相应 ascii 码作为写入 RAM 数据传出（对应输出 codeOut），这样就可以在 VGA 的 RAM 中写入相应的数据。考虑到时序问题，我们设置两个数组下标指针，一个是 waddr，指向当前屏幕上打印出来的最后一个字符（比如刚刚敲击‘E’键，waddr 即指向 E 在字符显存中的位置，也作为字符显存 RAM 中的写地址），另一个是 ptr，指向即将输入的位置（比如刚刚敲击‘E’键，ptr 即指向 E 字符所在位置的下一个位置）。两个指针均相对字符显存而言。在时序逻辑中，最开始（开机）或者清零端有效的时候将 waddr 和 ptr 全置 0。之后如果处理的键码属于需要在屏幕上输出字模的键码，那么将 waddr 的值置为 ptr 的值，然后 ptr 加 1，然后将字符显存 RAM 写使能置为 1。其他情况后面再详细考虑。

之后设计键盘的状态机。状态机的输出决定了字符显存写入的数据。状态机中设定一些标志位。其中 isEnd 表示键盘刚刚接收到 F0，isShift 表示 shift 键已经按下，delEnter 表示删除的时候如果删到某一行的头，那么再按一下删除应该

是删除回车键，之后再按一下才从上一行末尾开始删除。delEnter 表示已经删除了回车键。isE0 用作上下左右键的指示，这四个键的键码是一个 4 位的 16 进制数，前两位一定是 E0，因此为了判断是不是按下的上下左右键的键码，isE0 指示刚刚接收到的键码是 E0，表明后面跟这个的是上下左右的指示。

下面展示各种状态的名称和解释：

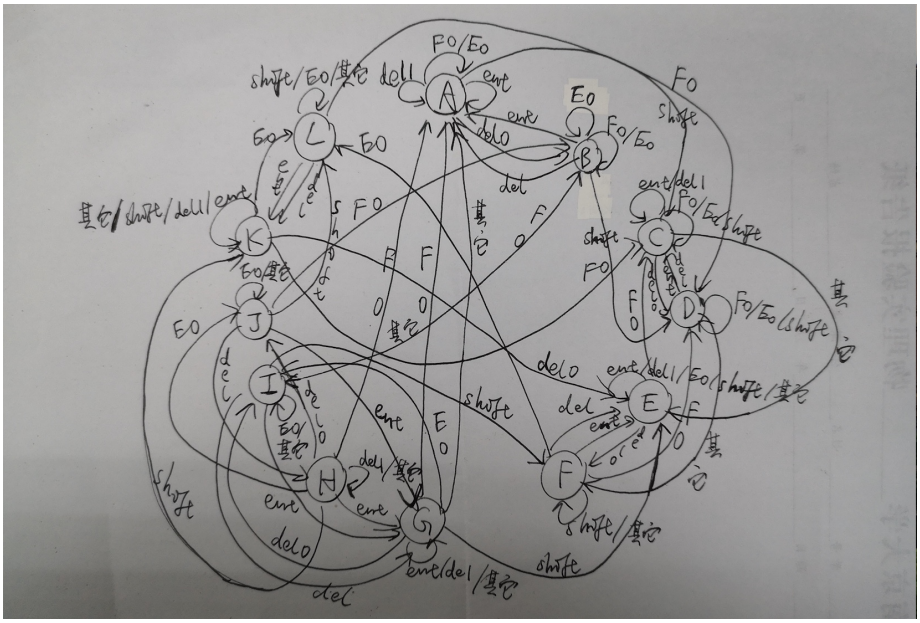
状态解释	isEnd	isShift	delEnter	isE0	状态名
接收到断码 F0（一个键松开），且 shift 没有按下	1	0	0	0	A
接收到断码 F0（一个键松开），且 shift 没有按下，且此刻回车键已经被删除	1	0	1	0	B
接收到断码 F0（一个键松开），且 shift 按下没有松开	1	1	0	0	C
接收到断码 F0（一个键松开），且 shift 按下没有松开，且此刻回车键已经被删除	1	1	1	0	D
接收到键码，且 shift 按下没有松开	0	1	0	0	E
接收到键码，且 shift 按下没有松开，且此刻回车键已经被删除	0	1	1	0	F
接收到键码，且 shift 没有按下	0	0	0	0	G
接收到键码 E0，且是上下左右键的开头指示	0	0	0	1	H
接收到键码，且 shift 没有按下，且此刻回车键已经被删除	0	0	1	0	I
接收到键码 E0，且是上下左右键的开头指示，且此刻回车键已经被删除	0	0	1	1	J
接收到键码 E0，且是上下左右键的开头指示，且 shift 按下没有松开	0	1	0	1	K
接收到键码 E0，且是上下左右键的开头指示，且 shift 按下没有松开，且此刻回车键已经被删除	0	1	1	1	L

下面是状态转移表：

		键码				
状态名	F0	回车	删除	E0	shift	其它
A	A	A	A/B	A	C	G
B	B	A	A	B	D	I
C	C	C	C/D	C	C	E
D	D	C	C	D	D	F
E	C	E	E/F	E	E	E
F	D	E	E	L	F	F
G	A	G	G/I	I	E	G
H	A	G	H/J	J	K	H
I	B	G	G	I	F	I
J	B	G	H	J	L	J
K	C	K	E/K	L	K	K
L	D	K	K	L	L	L

状态转移表中标红的有两种下一个状态的取决于当前的 ptr。如果当前的 ptr 在一行的开始，那么状态就转移到 delEnter 为 1 的地方；否则状态转移至自己。

状态转移图：



下面展示的是一些在键盘模块利用 assign 语句赋值的中间变量，这些变量在键盘接收数据的时候会被利用实现一些状态需要完成的功能，这些变量在后面都会解释：

```
assign waddr1=waddr%70;
assign ptr1=ptr%70; //这一行已经有多少个字符
assign ptr2=ptr+70-ptr1;
assign ptr3=ptr/70; //第几行的字符
assign ptr4=((ptr3-1) << 6)+((ptr3-1) << 2)+((ptr3-1) << 1)+back[ptr3-1]-1;
assign ptr5=((ptr3-1) << 6)+((ptr3-1) << 2)+((ptr3-1) << 1)+69;
assign waddr2=((ptr3-1) << 6)+((ptr3-1) << 2)+((ptr3-1) << 1)+back[ptr3-1]-1;
assign waddr3=((ptr3-1) << 6)+((ptr3-1) << 2)+((ptr3-1) << 1)+69;
```

下面详细讲解键盘模块的处理方法：

首先是初始化。然后是一个接收数据的模块，与实验八中相同，是将键盘发送的数据中有效的 12 位赋给 buffer 缓冲数据存储（最后还有一个 1，被抛掉）：

```
initial
begin
    data=0;
    count=0;
    isEnd=0;
    isShift=0;
    waddr=0;
    ptr=0;
    en=0;
    delEnter=0;
    isE0=0;
end
always @ (posedge clk)
begin
    ps2_clk_sync <= {ps2_clk_sync[1:0],ps2_clk};
end
wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];
always @ (posedge clk)
begin
    if(clrn==1)
    begin
        count<=0;
        buffer<=0;
    end
    else if(sampling)
    begin
        if(count == 4'd10)
        begin
            if((buffer[0] == 0) && (ps2_data) && (^buffer[9:1]))
                count <= 0;
        end
        else
        begin
            buffer[count] <= ps2_data;
            count <= count + 3'b1;
        end
    end
end
```

之后一个模块由 50MHz 时钟激励，是处理键码的主要部分。如果清零端有效，则将相应的标志位清回原始状态：

```
always @ (posedge clk)
begin
    if(clrn==1)
    begin
        isEnd<=0;
        isShift<=0;
        en<=0;
        ptr<=0;
        data<=0;
        waddr<=0;
        delEnter<=0;
        isE0<=0;
    end
```

之后是在 buffer 有效的时候根据具体接受的键码解释除了转移至相应的状态还需要进行的操作：

（1）如果接收到 F0，转移至相应状态（改变对应标志位即可）：

```
    else if((count==4'd10) && (sampling) && (buffer[0]==0) && (ps2_data) && (^buffer[9:1]))
    begin
        if(buffer[8:1]==8'b11110000)
        begin
            isEnd<=1;
            en<=0;
            isE0<=0;
        end
```

下面的（2）～（7）均是接收到键码且不是 F0 后面紧跟的键码（即 isEnd 为 0）时的情况：

（2）如果接收到的是 shift 键，将 shift 标志位置 1 并修改及其它标志位：

```
100         else
101         begin
102             if(isEnd==0)
103             begin
104                 if(buffer[8:1]==8'h12||buffer[8:1]==8'h59)
105                 begin
106                     isShift<=1;
107                     isE0<=0;
108                     delEnter<=0;
109                 end
```

（3）如果接收到退格键（del），那么分两种情况讨论。第一种情况是现在的光标（ptr）位置在不在某一行的开头。说明这一行还有字符，那么删除的话就删除这些字符。将此时的 waddr 作为写入地址，en 置为 1，写入字符显存 ascii 码设置为 00，表明将这个字符设置成打印出来的全黑，代表删除掉。如果 ptr 在某一行的开头，如果 delEnter（删回车）为 0，那么将其置为 1，表明删除回车键；如果 delEnter 为 1，说明已经删除了回车键，于是将光标挪到上一行的字符末端。这里为了存储每一行最后一个字符在什么位置，我们使用一个寄存器类型数组 back，记录了每一行有多少个字符。这个字符在删除的时候使用对应的行指针指向删除的字符。行指针就是临时变量中的 ptr3。在接受正常的字符（需要在屏幕上打印出来）时，这个数组对应行的字符数加 1，删除的时候每删一次（除删回车键），就将对应行的字符数量减 1。注意其中 waddr 指针赋值的时序问题，因此利用了一个 tempwaddr 指针：

```
110             else if(buffer[8:1]==8'h66) //退格键
111             begin
112                 if(back[ptr3]!=0) //这一行还有字符
113                 begin
114                     back[ptr3]<=back[ptr3]-1;
115                     ptr<=ptr-1;
116                     waddr<=tempwaddr;
117                     tempwaddr<=tempwaddr-1;
118                     data<=8'h00;
119                     en<=1;
120                 end
121                 else if(ptr3!=0) //这一行已经没有字符且不是第一行
122                 begin
123                     if(delEnter==0)
124                     begin
125                         delEnter<=1;
126                     end
127                     else
128                     begin
129                         back[ptr3-1]<=back[ptr3-1]-1;
130                         ptr<=ptr4;
131                         waddr<=waddr2;
132                         data<=8'h00;
133                         en<=1;
134                         delEnter<=0;
135                     end
136                 end
137                 isE0<=0;
138             end
```

（4）如果接收到的是回车键，那么就是换一行。如果在接收到回车键之前光标（ptr，下一个即将要写入的位置）正好在某一行的开头，即现在字符显存中正好一行是满的，那么这里的回车就相当于自动换行（等于 ptr 不变）。如果不是这一行还没有满，那么就进行换行的计算。ptr 指向下一行的起始位置。计算的

公式为 ptr=ptr+70-ptr%70。ptr%70 用 assign 语句以 ptr1 表示，ptr+70-ptr%70 用 assign 语句以 ptr2 表示：

```
139             else if(buffer[8:1]==8'h5A) //回车键
140             begin
141                 if(ptr!=0&&waddr1==69)
142                     ptr<=ptr;
143                 else
144                     ptr<=ptr2;
145                 en<=0;
146                 delEnter<=0;
147                 isE0<=0;
148             end
```

（5）如果接收到的是 E0 键，指示下面输入的将是以 E0 开头的由 4 个 16 进制组成的键码（包括上下左右键），这时将 isE0 信号置 1：

```
149             else if(buffer[8:1]==8'hE0)
150             begin
151                 isE0<=1;
152                 en<=0;
153             end
```

（6）如果接收到的键的键码为 0x75，0x6B，0x72，0x74，由于上下左右键的键码是这四个前面加上 E0，因此在接收到这四个键码的时候做判断，如果 isE0 指示位为 1，说明前面一个键码是 E0，表示现在输入的就是上下左右键。否则不作处理。这里 isE0 除了在（5）中将 isE0 置为 1 之外其余模块的最后都要将 isE0 置为 0（包括这个模块的结尾部分），从而可以保证如果 isE0 为 1，那么一定可以说明现在处理的键码的上一个键码就是 E0。上下左右键的实现在这里用了一个很巧妙的方法。我们不直接去修改字符显存，而是直接去修改 VGA 的扫描。也就是说，光标的显示是由 ptr 指针实现的，而上下左右的移动不过是改变了 ptr 的在字符显存中的位置（注意这里字符显存的写使能为 0，只是改变指针位置而不改变显存内容）。因此只需改变 ptr 的值，这样就可以实现光标移动但是不会在光标挪走之后原来光标覆盖的位置的原内容被改写的情况，因为字符显存没有修改。这样上下左右的实现就变成了单纯修改 ptr 指针的位置。根据一些计算将 ptr 赋为相应的值：

```
154             else if(buffer[8:1]==8'h75)
155             begin
156                 if(isE0==1)
157                 begin
158                     if(ptr3!=0)
159                     begin
160                         ptr<=ptr-70;
161                     end
162                     isE0<=0;
163                 end
164             end
165             else if(buffer[8:1]==8'h6B)
166             begin
167                 if(isE0==1)
168                 begin
169                     if(ptr!=0)
170                     begin
171                         ptr<=ptr-1;
172                     end
173                     isE0<=0;
174                 end
175             end
176             else if(buffer[8:1]==8'h72)
177             begin
178                 if(isE0==1)
179                 begin
180                     if(ptr3!=29)
181                     begin
182                         ptr<=ptr+70;
183                     end
184                     isE0<=0;
185                 end
186             end
187             else if(buffer[8:1]==8'h74)
188             begin
189                 if(isE0==1)
190                 begin
191                     ptr<=(ptr+1)%2100;
192                     isE0<=0;
193                 end
194             end
```

（7）接收到其余键码，也就是要在字符显存中写入的键码。这里将 data 赋为 buffer[8:1]，即有效键码的值，然后 waddr 指向 ptr，ptr 加 1。如果输入的 tab 键，那么 ptr 指针加 4，其余情况 ptr 指针加 1。每次的 waddr 是由上一次的 ptr 决定的，而 ptr 有指向了当前写入位置。因此时序逻辑正确。注意这里要将除 tab 键外的 en 置 1，表示字符显存写使能有效，写入 RAM。

```
195             else
196             begin
197                 data<=buffer[8:1];
198                 waddr<=ptr;
199                 tempwaddr<=ptr;
200                 back[ptr3]<=ptr1+1;
201                 if(buffer[8:1]==8'h0D)
202                 begin
203                     ptr<=(ptr+4)%2100;
204                     en<=0;
205                 end
206                 else
207                 begin
208                     ptr<=(ptr+1)%2100;
209                     en<=1;
210                 end
211                 delEnter<=0;
212                 isE0<=0;
213             end
214         end
```

（8）接收到的键码正好位于 F0 之后，如果是 shift 的键码，说明 shift 松开，将 isShift 置 0，修改其余标志位：

```
195             else
196             begin
197                 data<=buffer[8:1];
198                 waddr<=ptr;
199                 tempwaddr<=ptr;
200                 back[ptr3]<=ptr1+1;
201                 if(buffer[8:1]==8'h0D)
202                 begin
203                     ptr<=(ptr+4)%2100;
204                     en<=0;
205                 end
206                 else
207                 begin
208                     ptr<=(ptr+1)%2100;
209                     en<=1;
210                 end
211                 delEnter<=0;
212                 isE0<=0;
213             end
214         end
```

最后，利用 assign 给出本模块的输出。codeOut 是输出到字符显存模块中的 ascii 码，enOut 为输出到字符显存模块中的写使能，waddrOut 为显存写地址，ptrOut 为光标位置指示，ascii 为调试用输出，可忽略：

```
226     assign codeOut=asciicode;
227     assign ascii=back[ptr3];
228     assign enOut=en;
229     assign waddrOut=waddr;
230     assign ptrOut=ptr;
231     endmodule
```

注意上面有一个 ascii 码的输出。键盘模块里面并没有键码 zhuanascii 码的操作。这个操作在另外一个模块 codeChange 中展现并且被键盘模块时刻调用，也就是根据 data 的值一直给出其对应的 ascii 码：

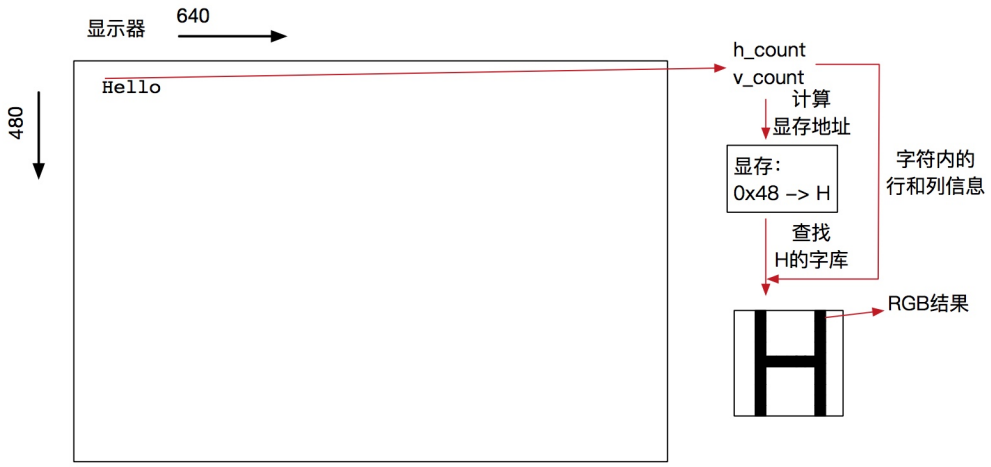
```
77      codeChange func(.code(data),.shift(isShift),.outdata(asciicode)); //发送的键码时刻转化为ASCII码
```

codeChange 模块利用 case 语句判断键码，data 就是输入的键码，shift 表示是否按下 shift 键，由键盘模块传入，outdata 为转化后的 ascii 码。shift 按下和不按下是两套转化方式（只展示部分）：

```
1   module codeChange(code,shift,outdata);
2   input [7:0]code;
3   input shift;
4   reg [7:0]data;
5   output [7:0]outdata;
6   always @ (*)
7   begin
8       if(shift==0)
9       begin
10          case(code)
11              8'h01:data=8'h5f;
12              8'h76:data=8'h1b;
13              8'h05:data=8'h70;
14              8'h06:data=8'h71;
15              8'h04:data=8'h72;
16              8'h0c:data=8'h73;
17              8'h03:data=8'h74;
18              8'h0b:data=8'h75;
19              8'h83:data=8'h76;
20              8'h0a:data=8'h77;
21              8'h09:data=8'h79;
22              8'h78:data=8'h7a;
23              8'h07:data=8'h7b;
75              8'h4a:data=8'h2f;
76              8'h59:data=8'h10;
77              8'h14:data=8'h11;
78              8'h11:data=8'h12;
79              8'h29:data=8'h20;
80              default:data=8'h00;
81          endcase
82      end
83      else
84      begin
85          case(code)
86              8'h01:data=8'h5f;
87              8'h76:data=8'h1b;
88              8'h05:data=8'h70;
89              8'h06:data=8'h71;
90              8'h04:data=8'h72;
91              8'h0c:data=8'h73;
152             8'h14:data=8'h11;
153             8'h11:data=8'h12;
154             8'h29:data=8'h20;
155             default:data=8'h00;
156         endcase
157     end
158 end
159 assign outdata=data;
160 endmodule
```

这样键盘模块的设计基本就完成了。

最后是顶层实体对于 VGA 和键盘的调用以实现交互。顶层实体的核心功能是从 VGA 模块拿到扫描的行列信息，根据此信息将该行该列映射到字符显存中对应的元素，取出该元素的 ascii 码，利用该 ascii 码查到字模缓存中该字符的颜色信号编码，再根据行列信息与此编码一一对应配置 vga_data 的值，重新传回 VGA 模块：



下面展示顶层实体模块中所有的输入输出以及利用的中间变量：

```
module lab11(CLK_50,reset,CLK_25,CLK_PS2,PS2_DAT,hsync,vsync,blank,vga_r,vga_g,vga_b,vga_sync,ledA,ledB);
input CLK_50;
input reset;
input CLK_PS2;
input PS2_DAT;
reg [23:0] vga_data;
reg [25:0] clockcount;
wire [11:0] waddr;
wire [11:0] ptr;
wire [9:0] ptr1;
wire [9:0] ptr2;
wire [9:0] h_addr;
wire [9:0] v_addr;
wire [7:0] asciicode;
wire [7:0] vgaasciicode;
wire [7:0] ascii;
wire [11:0] addr;
wire [9:0] h_addr1;
wire [9:0] v_addr1;
wire [9:0] h_addr2;
wire [9:0] v_addr2;
wire [11:0] lineaddr;
wire clear;
wire en;
reg [11:0] text [4095:0];
reg [11:0] line;
reg dot;
reg point;
output CLK_25;
output hsync;
output vsync;
output blank;
output [7:0] vga_r;
output [7:0] vga_g;
output [7:0] vga_b;
output reg vga_sync;
output [6:0]ledA;
output [6:0]ledB;
```

输入如下：CLK_50 表示开发板 50MHz 时钟，reset 为重置键，高电平有效，该信号作为所有设置清零或重置端模块的信号。CLK_PS2 是键盘发送的时钟，供键盘模块使用，CLK_PS2 为接受键盘发送的数据，供键盘模块使用。输出 CLK_25（VGA 时钟），hysnc、vysnc（同步信号），blank（消隐信号），vga_r、vga_g、vga_b（颜色信号），vga_sync（VGA 同步信号，长期置零）。这些输出均输出到开发板上 VGA 控制芯片的对应引脚，与实验九相同。ledA 和 ledB 为调试用输出，在此忽略。顶层实体中用到的中间变量后面会讲到。

在顶层实体中，设定一个每个元素长度为 12 位，一共有 4096 个元素的寄存器类型数组 text，作为字符显存。也就是给出的 ascii 码对应的.txt 文本文件中的数据，利用$readmemh 从.txt 文件中读取数据，填充 text。text 每 16 行的的元素（一个元素占一行，一行 12 位）对应一个字符的点阵信息。

```
39    assign vag_sync=1'b0;
40    assign clear=~reset;
41    assign h_addr1=h_addr/9;
42    assign h_addr2=(h_addr%9)-1;
43    assign v_addr1=v_addr/16;
44    assign v_addr2=v_addr%16;
45    assign addr=(v_addr1 << 6)+(v_addr1 << 2)+(v_addr1 << 1)+h_addr1;
46    assign lineaddr=(vgaasciicode << 4)+v_addr2;
47    assign ptr1=ptr/70;
48    assign ptr2=ptr%70;
49    initial
50  begin
51      $readmemh("vga_font.txt",text,0,4095);
52      point=1;
53      clockcount=0;
54    end
55
56    clkgen #(25000000) my_vgaclk(CLK_50,clear,1'b1,CLK_25);
```

这里 clear 传入所有存在清零或重置信号的模块。由于输入 reset 是一个 button 类型，按下去（低电平有效），因此做一个取反操作，clear 转化为高电平有效。v_addr1 和 h_addr1 指示扫描点行列 v_addr、h_addr 对应到字符显存的那个 ascii 码位置，作为键盘模块的参数。v_addr2 和 h_addr2 是对于找到的 ascii 码，该 ascii 码的 16x9 点阵的每个位置的指示，用来确定该位置的颜色是白色还是黑色。addr 是作为字符显存查询用的具体地址，利用 v_addr1 和 h_addr1 计算，lineaddr 指示找到的 ascii 码的对应的字模缓存的特定行，取出 12 位的一行的颜色信息，利用 vga 模块查询 IP 和字符显存之后返回的 ascii 码值以及 v_addr2 计算。利用逻辑左移的目的是避免乘法带来的延迟。ptr1 和 ptr2 指示字符显存中光标位置，用于后面的光标显示。clockcount 是计数器，用于光标闪烁。通过 initial 读取文件填充字模缓存，调用 clkgen 生成 25MHz 时钟 CLK_25。之后调用 vga 模块和键盘模块：

```
68    newvga_ctrl func1(
69        .pclk(CLK_25),
70        .reset(clear),
71        .clk50(CLK_50),
72        .clk25(CLK_25),
73        .en(en),
74        .waddr(waddr),
75        .raddr(addr),
76        .asciicode(asciicode),
77        .vga_data(vga_data),
78        .h_addr(h_addr),
79        .v_addr(v_addr),
80        .hsync(hsync),
81        .vsync(vsync),
82        .valid(blank),
83        .vga_r(vga_r),
84        .vga_g(vga_g),
85        .vga_b(vga_b),
86        .vgaasciicode(vgaasciicode)
87    );
88
89    newkeyboard func2(
90        .clk(CLK_50),
91        .clrn(clear),
92        .ps2_clk(CLK_PS2),
93        .ps2_data(PS2_DAT),
94        .enOut(en),
95        .waddrOut(waddr),
96        .codeOut(asciicode),
97        .ascii(ascii),
98        .ptrOut(ptr)
99    );
```

可以看到调用 vga 模块，传入相应的参数中，en 作为 wire 类型由键盘模块传出，传入 vga 模块，表示字符显存写使能；waddr 作为 wire 类型由键盘模块传出，传入 vga 模块，表示字符显存写地址；addr 即为本模块实时计算结果作为字符显存读地址，vgaasciicode 即为根据读地址读到的数据。另外键盘模块中的输出 ptr 是光标位置指示，ascii 为调试用，可忽略。

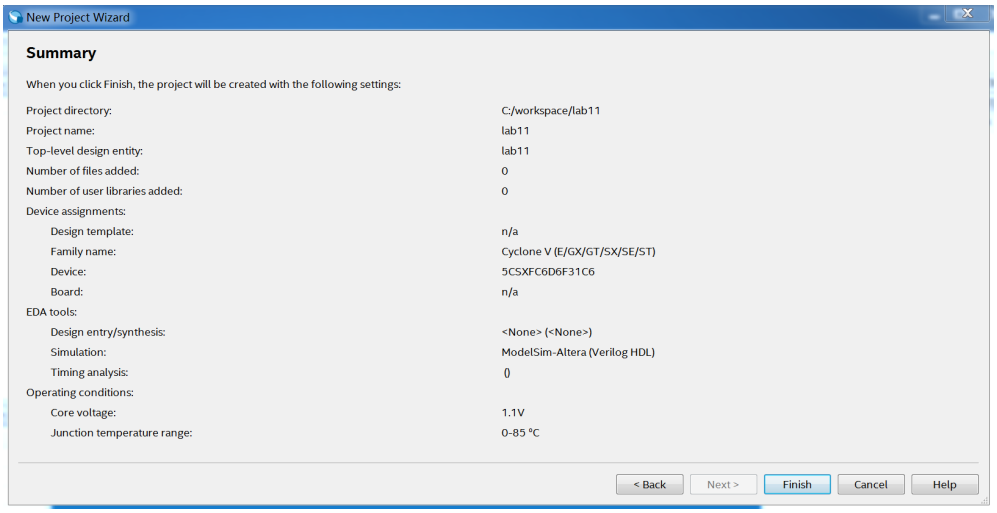
最后是核心的修改 vga_data 从而实现屏幕输出的语句块。利用 50MHz 激励，当扫描到的 v_addr 和 h_addr 处于光标位置时，就将该 16x9 块的低三行设置为白色，其余为黑色。其他情况按照查询字符显存获得 ascii 码，根据该 ascii 码对应的字模缓存的 16x9 具体每行每列每个点的信号是 1（黑）还是 0（白）来设置 vga_data。时钟激励一次 clockcount+1，循环计数，clockcount 小于 25000000 时光标为白色，其余为黑色，这样就可以看到光标以 1s 的周期闪烁。point 是为了打印光标的时候避免一些奇怪错误设置的标志位，如果该位置是光标，那么就不进行读字模缓存的判断（即执行 else if 语句）：

```
101     always @ (posedge CLK_50)
102   begin
103       line<=text[lineaddr];
104       if(v_addr1==ptr1&&h_addr1==ptr2)
105       begin
106           point<=0;
107           if(v_addr2>12&&clockcount<=25000000)
108               vga_data<=24'hFFFFFF;
109           else
110               vga_data<=24'h000000;
111       end
112       else if(line[h_addr2]&&h_addr1<70&&point)
113           vga_data<=24'hFFFFFF;
114       else
115       begin
116           vga_data<=24'h000000;
117           point<=1;
118       end
119       clockcount<=(clockcount+1)%50000000;
120   end
121
122   endmodule
```

这样，整个工程的设计基本上就完成了。

5、实验流程

首先新建一个工程：



之后新建.v 文件，编写逻辑代码（代码创建流程见上面）

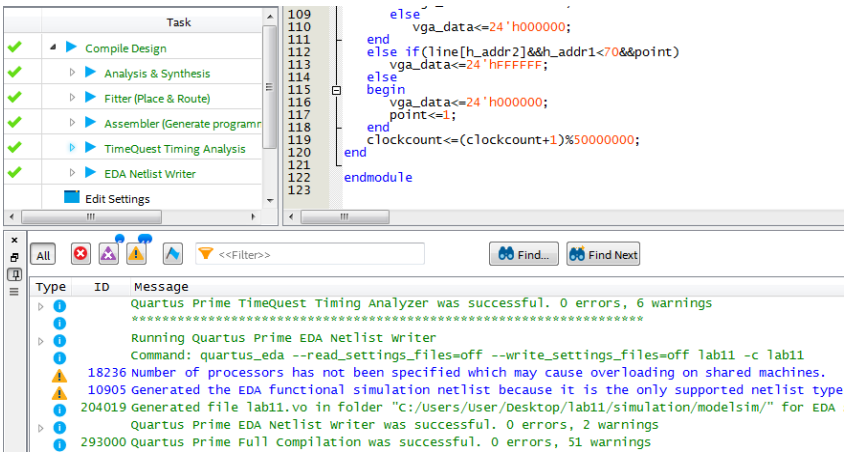
进行规划引脚的操作（这里不展示调试用的输出的引脚规划）：

CLK_25	Output	PIN_AK21	4A	B4A_N0	PIN_AK21	2.5 V
CLK_50	Input	PIN_AF14	3B	B3B_N0	PIN_AF14	2.5 V
CLK_PS2	Input	PIN_AB25	5A	B5A_N0	PIN_AB25	2.5 V
PS2_DAT	Input	PIN_AA25	5A	B5A_N0	PIN_AA25	2.5 V
blank	Output	PIN_AK22	4A	B4A_N0	PIN_AK22	2.5 V
hsync	Output	PIN_AK19	4A	B4A_N0	PIN_AK19	2.5 V

reset	Input	PIN_AJ4	3B	B3B_N0	PIN_AJ4	2.5 V
vga_b[7]	Output	PIN_AK16	4A	B4A_N0	PIN_AK16	2.5 V
vga_b[6]	Output	PIN_AJ16	4A	B4A_N0	PIN_AJ16	2.5 V
vga_b[5]	Output	PIN_AJ17	4A	B4A_N0	PIN_AJ17	2.5 V
vga_b[4]	Output	PIN_AH19	4A	B4A_N0	PIN_AH19	2.5 V
vga_b[3]	Output	PIN_AJ19	4A	B4A_N0	PIN_AJ19	2.5 V
vga_b[2]	Output	PIN_AH20	4A	B4A_N0	PIN_AH20	2.5 V
vga_b[1]	Output	PIN_AJ20	4A	B4A_N0	PIN_AJ20	2.5 V
vga_b[0]	Output	PIN_AJ21	4A	B4A_N0	PIN_AJ21	2.5 V
vga_g[7]	Output	PIN_AH23	4A	B4A_N0	PIN_AH23	2.5 V
vga_g[6]	Output	PIN_AK23	4A	B4A_N0	PIN_AK23	2.5 V
vga_g[5]	Output	PIN_AH24	4A	B4A_N0	PIN_AH24	2.5 V
vga_g[4]	Output	PIN_AJ24	4A	B4A_N0	PIN_AJ24	2.5 V
vga_g[3]	Output	PIN_AK24	4A	B4A_N0	PIN_AK24	2.5 V
vga_g[2]	Output	PIN_AH25	4A	B4A_N0	PIN_AH25	2.5 V
vga_g[1]	Output	PIN_AJ25	4A	B4A_N0	PIN_AJ25	2.5 V
vga_g[0]	Output	PIN_AK26	4A	B4A_N0	PIN_AK26	2.5 V
vga_r[7]	Output	PIN_AJ26	4A	B4A_N0	PIN_AJ26	2.5 V
vga_r[6]	Output	PIN_AG26	4A	B4A_N0	PIN_AG26	2.5 V
vga_r[5]	Output	PIN_AF26	4A	B4A_N0	PIN_AF26	2.5 V
vga_r[4]	Output	PIN_AH27	4A	B4A_N0	PIN_AH27	2.5 V
vga_r[3]	Output	PIN_AJ27	4A	B4A_N0	PIN_AJ27	2.5 V
vga_r[2]	Output	PIN_AK27	4A	B4A_N0	PIN_AK27	2.5 V
vga_r[1]	Output	PIN_AK28	4A	B4A_N0	PIN_AK28	2.5 V
vga_r[0]	Output	PIN_AK29	4A	B4A_N0	PIN_AK29	2.5 V
vga_sync	Output	PIN_AJ22	4A	B4A_N0	PIN_AJ22	2.5 V
vsync	Output	PIN_AK18	4A	B4A_N0	PIN_AK18	2.5 V
<<new node>>						

其中 CLK_25 接开发板上的 VGA_CLK，blank 接 VGA_BLANK_N，CLK_50 接系统 50MHz 时钟，reset 对应 SW[0]，hsync、vsync 接 VGA_HS、VGA_VS，vga_sync 接 VGA_SYNC_N，vga_r、vga_g、vga_b 数组对应 VGA_R、VGA_G、VGA_B 数组。CLK_PS2 和 CLK_DAT 接键盘的时钟输入和数据输入。

之后编译通过，生成二进制文件 lab11.sof，将二进制文件写入开发板验证功能：

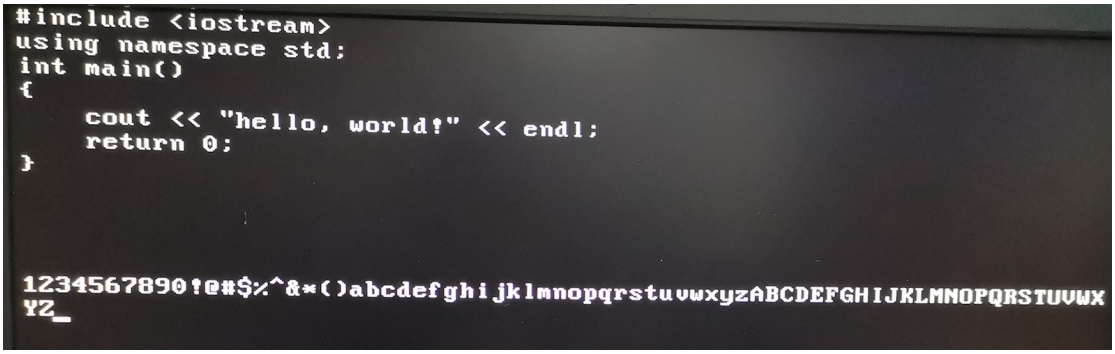


6、测试方法

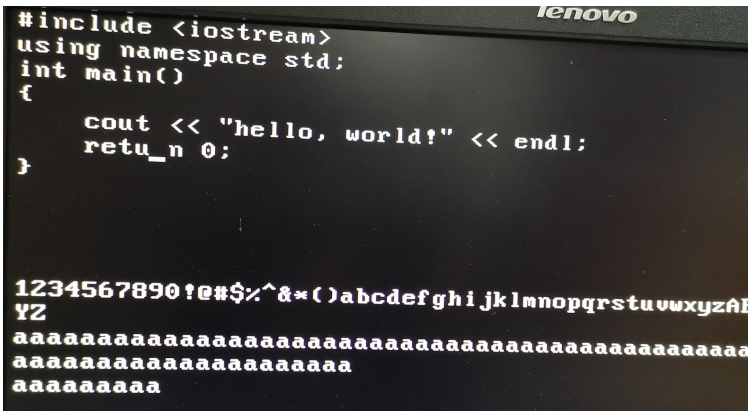
开发板接显示器直接验证。

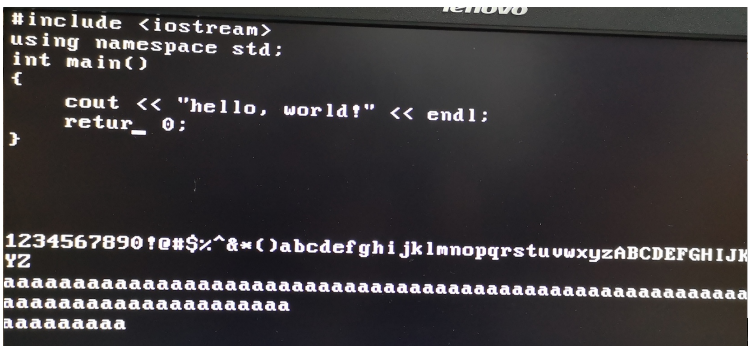
7、实验结果

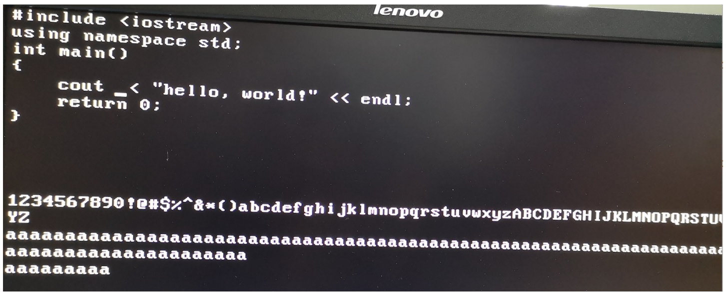
（1）输出所有的字符，包括 tab 键功能（图中 cout 前面为 tab 键按下后显示的功能），shift 键按下之后显示特殊字符（如大写字符）功能，一直按下一直输出，松开键停止输出，换行（支持连续换多行），字符后面的光标，且光标可以闪烁：

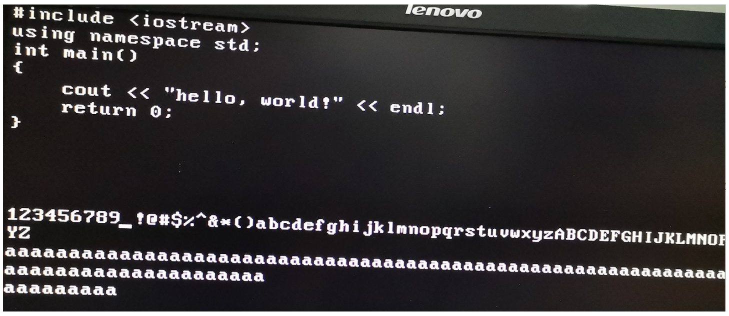


（2）支持光标的上下左右移动，可以在移动后指向的位置直接输入也可以继续移动光标，且光标移动走之后不会改变原来位置的字符显示：

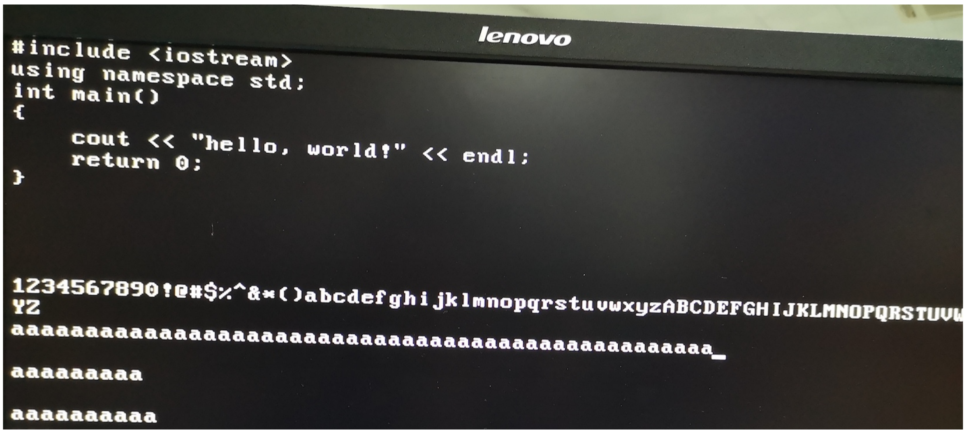








（3）支持删除，且可以跨行删除，即删除到一行开头之后还可以继续删除上一行内容：



8、实验中遇到的问题及解决办法

（1）一开始设计字符显存的时候用的是寄存器类型的数组，并且在时序逻辑中对于这个字模显存进行内容的修改。也就是说，当接受的键码需要写到字符显存中的时候，将键码转化后的 ascii 码写入字符显存对应地址中：

```
        if(isEnd==0)
        begin
            if(buffer[8:1]==8'h12||buffer[8:1]==8'h59)
                isShift<=1;
            else
            begin
                data<=buffer[8:1];
                ptr<=(ptr+1)%2100;
                rom[ptr]<=asciicode;
                codeOut<=rom[70*lin_addr+col_addr];
            end
        end
        else
        begin
            if(buffer[8:1]==8'h12||buffer[8:1]==8'h59)
                isShift<=0;
            isEnd<=0;
        end
```

上图中 rom 即为字符显存，codeOut 为传出的扫描地址对应的 ascii 码。这样写出来的工程在编译的时候超过了十分钟，而且在时序上出现了严重的错误，按下一个键之后再按一个键才会输出第一次按下的那个键，即每次按下一个键屏幕上打印出来的是上一次按下的键，且字符的颜色也不是纯白色。这里原因主要是键盘时序的错误，对于字符显存利用同等频率的时钟读和写，可能导致未写入的时候已经读出以及对于寄存器型数组元素很多的时候要频繁地进行写和读操作，导致开发板耗费大量时间编译。后来改成了利用 IP 核的 RAM 实现的字符显存，利用比读驱动时钟快一倍的写时钟进行数据写入，这样不仅编译时间缩短到一分钟，而且时序上不会发生错误。

（2）一开始在顶层实体中改变 vga_data 的 always 语句块是用 25MHz 时钟激励的，在屏幕上验证的时候发现每次输出会打印一些额外的东西，即输出的每个字符的右侧还带有一些白色的像素点。后来改为 50MHz 时钟激励就正常输出了，推测与写入 IP 核的 50MHz 写时钟有关。

（3）一开始在顶层实体里调用键盘模块和 vga 模块的时候有一些传入的参数写错了，有的参数甚至还没有在前面定义，但是编译通过，结果出现了莫名其妙的错误，后来的检查的过程中发现了这些变量还没有定义，或者是长度写错了（比如本来是 12 位的变量但是只定义了 1 位），这些编译的时候都没有报错。之后修正了这些错误，问题解决。

9、实验中得到的启示

本次实验是要求我们充分利用之前学到的知识，将之前各种实现整合到一起，实现一个模块之间的交互。这要求我们充分掌握之前学过的知识并加以运用。另外，将一个大问题分化成几个小问题也是这次实验考察的重点。遇到一个问题的时候，要尝试将其分成数个小问题，比如这次的交互界面就可以分成键盘模块和 vga 模块以及交互模块，键盘模块又可以分为处理键码模块以及键码转 ascii 码模块，等等，之后分步将这些小问题一个一个解决，最后理清这些小问题与大问题之间的关系，将解决的小问题整合起来，就解决了大问题。此外，对于向本次实验这样很难进行软件模拟的工程，可以通过在添加开发板上的输出来做调试，比如用七段数码管来显示当前接收到的键码等等。还有，出现问题的时候，要学会分模块调试，利用控制变量法去寻找出错的地方。对于一些无法理解的错误，可以通过修改参数（比如修改常量）在开发板上尝试的方法来进行调试。

10、意见和建议：暂时还没有。