实验十一

字符输入界面

实验报告　　日期：2020 年 12 月 1 日

姓名： 朱嘉琦

学号： 191220185

班级： 数电实验一班

邮箱： 1477194584@qq.com

实验十一报告—— 字符输入界面

191220185 朱嘉琦

一、实验目的

本实验将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

字符显示——字模

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，如图 11-1所示。

图 11-1: ASCII 字符字模

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。例如图11-2就是字模"A"与储存器的关系。

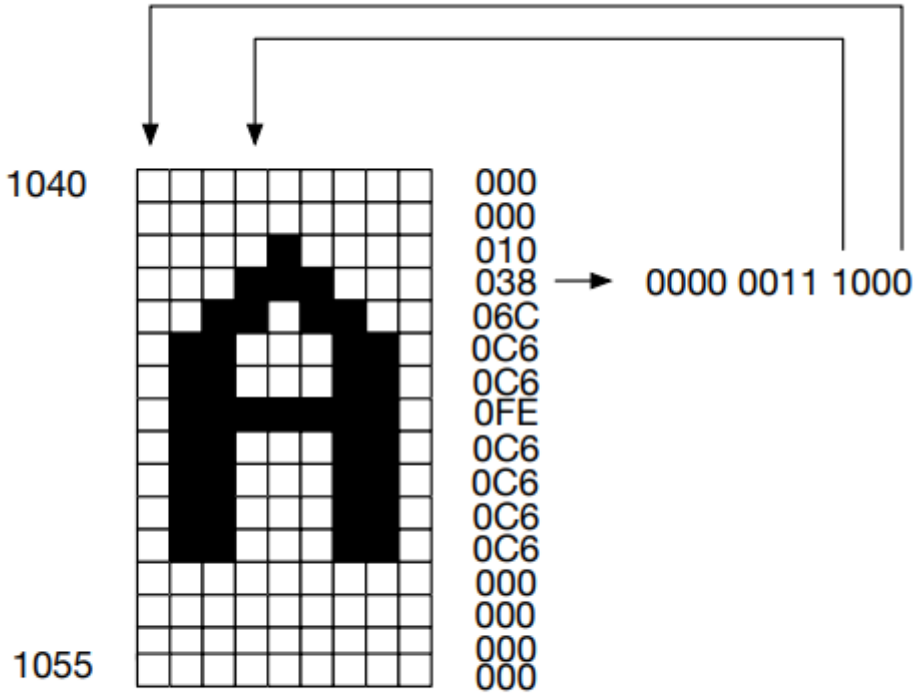


图 11-2: 字模“A”与存储器的关系

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，再根据扫描线的行和列信息，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。（如图11-3所示）

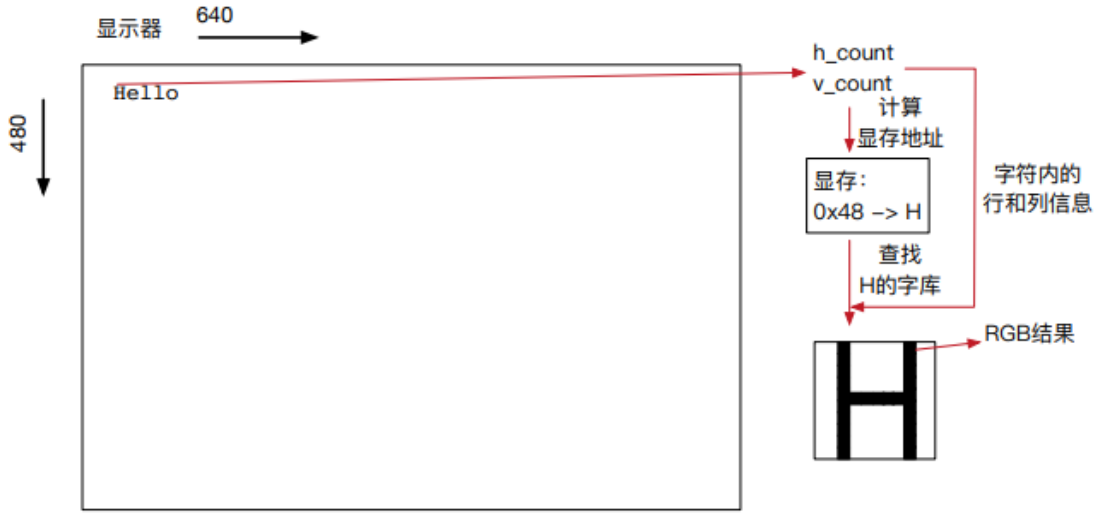


图 11-3: 字符显示流程示意图

因此，将显示的**过程**如下：

1. 根据当前扫描位置，获取对应的字符的 x,y 坐标，以及扫描到单个字符点阵内的行列信息。
2. 根据字符的 x,y 坐标，查询字符显存，获取对应 ASCII 编码。

3. 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。
4. 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其他彩色字符，只需要按自己的需求分别设置背景颜色和字符颜色即可。

三、实验环境/器材

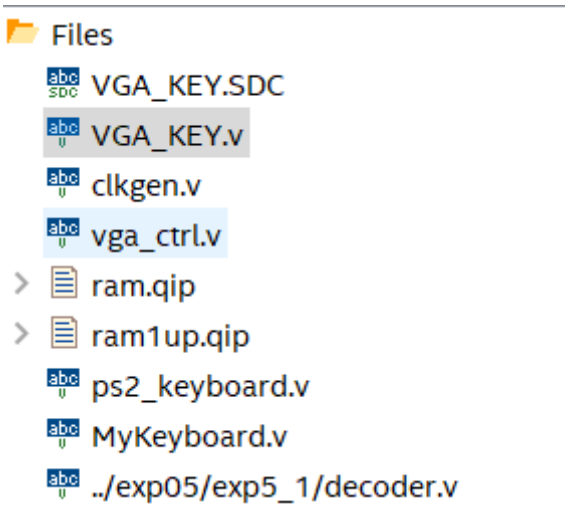
实验环境是Quarters 17.1 Lite，实验器材是DE10 开发板

四、程序代码或流程图

1. 工程文件

本次实验涉及到多个模块

VGA_KEY



2. VGA控制模块

1）vga_ctrl.v

VGA控制模块与显示图片实验保持一致，所以在此仅给出接口声明

```
module vga_ctrl(
    input pclk, //25MHz时钟
    input reset, //置位
    input [11:0] vga_data, //上层模块提供的VGA颜色数据
    output [9:0] h_addr, //提供给上层模块的当前扫描像素点坐标
    output [9:0] v_addr,
    output hsync, //行同步和列同步信号
    output vsync,
    output valid, //消隐
    output [3:0] vga_r, //RGB
    output [3:0] vga_g,
    output [3:0] vga_b
    );
endmodule
```

2) clkgen.v

```
module clkgen(
    input clkin,
    input rst,
    input clken,
    output reg clkout
    );
    parameter clk_freq=1000;
    parameter countlimit=50000000/2/clk_freq; // count

    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
```

3. 键盘模块

1）ps2_keyboard.v

与之前实验的ps2_keyboard.v保持一致，仅给出接口声明

```
module ps2_keyboard(clk,clrn,ps2_clk,ps2_data,data,ready,nextdata_n,overflow);
    input clk,clrn,ps2_clk,ps2_data;
    input nextdata_n;
    output [7:0] data;
    output reg ready;
    output reg overflow; // fifo overflow
endmodule
```

2) MyKeyboard.v

与之前的MyKeyboard.v基本保持一致，多给出三个output分别表示ascii码值，大写ascii码值和键码data值，接口声明如下：

```
module MyKeyboard(input clk,
                  input clrn,
                  input ps2_clk,
                  input ps2_data,
                  output reg [7:0] data,
                  output [7:0] ascii,
                  output [7:0] ascii_u,
                  output [6:0] d_digit_low,
                  output [6:0] d_digit_high,
                  output reg [6:0] a_digit_low,
                  output reg [6:0] a_digit_high,
                  output [6:0] c_digit_low,
                  output [6:0] c_digit_high,
                  //output reg[15:0] count,
                  output reg ctrl,
                  output reg shift,
                  output reg up,
                  output reg caps
                  );
endmodule
```

4. 顶层模块

```
//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module VGA_KEY(

    ////////////// CLOCK //////////
    input                       CLOCK2_50,
    input                       CLOCK3_50,
    input                       CLOCK4_50,
    input                       CLOCK_50,

    ////////////// KEY //////////
    input             [3:0]     KEY,

    ////////////// SW //////////
    input             [9:0]     SW,

    ////////////// LED //////////
    output            [9:0]     LEDR,

    ////////////// Seg7 //////////
    output            [6:0]     HEX0,
    output            [6:0]     HEX1,
    output            [6:0]     HEX2,
    output            [6:0]     HEX3,
    output            [6:0]     HEX4,
    output            [6:0]     HEX5,

    ////////////// VGA //////////
    output                      VGA_BLANK_N,
    output            [7:0]     VGA_B,
    output                      VGA_CLK,
    output            [7:0]     VGA_G,
```

```
    output                          VGA_HS,
    output          [7:0]           VGA_R,
    output                          VGA_SYNC_N,
    output                          VGA_VS,

    ////////////// PS2 /////////
    inout                           PS2_CLK,
    inout                           PS2_CLK2,
    inout                           PS2_DAT,
    inout                           PS2_DAT2
);



//=======================================================
//  REG/WIRE declarations
//=======================================================
    assign VGA_SYNC_N=0;

    wire clk_key;
    wire clk_twink;
    wire[7:0]data;
    //wire[7:0]lastdata;
    wire[7:0]ascii_val;
    wire[7:0]ascii_val_u;

    wire[11:0]r;//row
    wire[11:0]c;//col
    wire[3:0]x;
    wire[3:0]y;

    reg[11:0]rom[4097:0];//存储点阵
    reg[11:0]cnt;//显示位置
    reg[11:0]cnt_tmp;
    reg[7:0]ascii[2129:0];//字符的ascii码

    reg[7:0]ascii_temp;
    wire [11:0]rom_temp;

    wire[9:0]h_addr;
    wire[9:0]v_addr;
    wire [11:0]vga_data;

    wire ctrl;
    wire shift;
    wire up;
    wire caps;

    reg flag=0;

    initial begin
        //ctrl=0;
        //shift=0;
        //up=0;
        //caps=0;
        integer i;
        for(i=0;i<=2129;i=i+1)
            ascii[i]=32;
```

```
        cnt=0;

$readmemh("E:/zjq/digital_circuit/design/exp11/vga_font.txt",rom,0,4097);
    end
//=============================================================
//  Structural coding
//=============================================================
    assign LEDR[0]=ctrl;
    assign LEDR[1]=shift;
    assign LEDR[2]=caps;
    assign LEDR[3]=up;

    clkgen #(25000000) my_vgaclk(CLOCK_50,1'b0,1'b1,VGA_CLK);
    clkgen #(15) my_keyclk(CLOCK2_50,1'b0,1'b1,clk_key);
    //clkgen #(2) my_twinkclk(CLOCK3_50,1'b0,1'b1,clk_twink);

    vga_ctrl my_vga(
    .pclk(VGA_CLK), //25MHz时钟
    .reset(1'b0), //置位
    .vga_data(vga_data), //上层模块提供的VGA颜色数据
    .h_addr(h_addr), //提供给上层模块的当前扫描像素点坐标
    .v_addr(v_addr),
    .hsync(VGA_HS), //行同步和列同步信号
    .vsync(VGA_VS),
    .valid(VGA_BLANK_N), //消隐
    .vga_r(VGA_R[7:4]), //RGB
    .vga_g(VGA_G[7:4]),
    .vga_b(VGA_B[7:4])
    );

    MyKeyboard(
    .clk(CLOCK_50),
    .clrn(SW[0]),
    .ps2_clk(PS2_CLK),
    .ps2_data(PS2_DAT),
    .data(data),
    .ascii(ascii_val),
    .ascii_u(ascii_val_u),
    .d_digit_low(HEX0),
    .d_digit_high(HEX1),
    .a_digit_low(HEX2),
    .a_digit_high(HEX3),
    .c_digit_low(HEX4),
    .c_digit_high(HEX5),
    //output reg[15:0] count,
    .ctrl(ctrl),
    .shift(shift),
    .up(up),
    .caps(caps)
    );

    always @ (posedge clk_key) begin
        if(flag==1)begin
            if(ascii[cnt]==95)
                ascii[cnt]<=32;
            else
                ascii[cnt]<=95;
        end
```

```
    if (data!=8'hF0)
    begin
        flag=1;
        if (data==8'h66) //backspace
            begin
                if(cnt%71==0)begin//begin of a row delete last n space
                    integer j;
                    for(j=cnt-1;(j>=cnt-71)&&(ascii[j]!=32);j=j-1);

                    ascii[cnt]<=32;//空格
                    ascii[cnt+1]<=32;//光标删除
                    cnt<=j;
                end
                else begin
                    cnt<=cnt-1;
                    ascii[cnt]<=32;//空格
                    ascii[cnt+1]<=32;//光标删除
                    //ascii[cnt+2]<=32;
                end
            end
        else if (data==8'h5A)//enter
            begin
                ascii[cnt]<=32;//空格
                //integer i=cnt;
                cnt<=(cnt/71+1)*71;//行数+1
                //put space into last row
                //for(;i<cnt;i=i+1)
                    //ascii[i]<=32;
            end
        else
            begin
                if(ctrl!=1&&caps!=1) begin
                    if(up==1)begin
                        if(shift==1)begin
                            if(data!=8'h12&&data!=8'h59)begin
                            ascii[cnt]<=ascii_val_u;
                            cnt=cnt+1;
                            end
                        end
                        else begin
                            ascii[cnt]<=ascii_val_u;
                            cnt=cnt+1;
                            end
                    end
                    else begin
                        ascii[cnt]<=ascii_val;
                        cnt=cnt+1;
                        end


                end
            end
    end
end

/*always @ (posedge clk_twink) begin //0x5f(95)
    if(ascii[cnt+1]==95)
```

```
            ascii[cnt+1]<=32;
        else
            ascii[cnt+1]<=95;
    end*/

    //计算c,r,x,y 得出字符显示的点阵位置
    assign c=h_addr/9;
    assign r=v_addr>>4;

    assign x=v_addr-(r<<4);
    assign y=h_addr-(c<<3+c);

    /*
    assign c=h_addr/9;
    assign r=v_addr>>4;
    assign x=v_addr%16;
    assign y=h_addr%9;*/

    always @(negedge VGA_CLK)
    begin
        ascii_temp<=ascii[r*71+c];
    end

    assign rom_temp=rom[{ascii_temp,x}];
    assign vga_data=(rom_temp[y]==1)?24'hFFFFFF:24'h000000;

endmodule
```

五、实验步骤

- 根据实验讲义学习字符通过VGA显示的原理，理解vga_font字模的表示，以及每个字模16*9显示的含义。
- 利用system builder同时接入vga和ps/2接口，实例化之前实验做得vga_ctrl控制模块和MyKeyboard键盘模块，对键盘模块稍加修改，使其将不同情况以及字符的ascii码作为output。
- 设计两个显存，分别表示所有ascii码所对应的字符显存，和VGA显示器上30*70的字符阵列显存，将vga_font.txt初始化到字符显存，再将字符阵列显存初始化为32（空格）。
- 由于本实验中需要用到多个不同时序，所以讲clk进行分频，分别为频率较高的VGA_CLK和频率较低的clk_key，因为键盘如果刷新率过高可能回导致误触，所以将频率设为15Hz。
- 通过键盘输入给vga赋值

```
ascii[cnt]<=ascii_val;
cnt=cnt+1;
```

- 根据h_addr和v_addr计算出其在ascii[]中的位置和对应的rom_temp的位置

```
//计算c,r,x,y 得出字符显示的点阵位置
    assign c=h_addr/9;
    assign r=v_addr>>4;

    assign x=v_addr-(r<<4);
    assign y=h_addr-(c<<3+c);
```

```
always @(negedge VGA_CLK)
    begin
        ascii_temp<=ascii[r*71+c];
    end

    assign rom_temp=rom[{ascii_temp,x}];
    assign vga_data=(rom_temp[y]==1)?24'hFFFFFF:24'h000000;
```

- 检查是否有语法错误，最终编译通过，连接到FPGA实验板上，通过键盘输入发现已经可以打字，接着再改写扩展其他功能。
- 在键盘输入上功能键虽然也有键码输出，但不应该在屏幕上显示，所以检测到这些键码时单独处理，由此实现backspace,enter,shift,caps lock,shift,ctrl等键，并且在当前输入的最后一个字符后实现一个随着时钟闪烁的光标。
- 检查是否有语法错误，最终编译通过，连接到FPGA实验板上，硬件验证。

六、测试方法

1. 将键盘和显示器连接到FPGA，编译后将sof文件导入开发板中实际测试电路，进行硬件验证能否正常打字。
2. 最后在代码中添加功能，实现光标，回格，大小写，换行等功能。

七、实验结果——字符输入界面

用键盘输入并且在显示屏上显示正确的字符，字符没有闪烁，边缘清晰，底色为黑色，字符为白色。同时支持shift caps lock控制大小写，有光标指示当前输入位置，可以使用backspace删除前序字符，可以用enter换行等等。（完成5个高级功能并且已经给助教验收）

八、思考

为了方便指针操作，我直接在程序中写了一个二维数组作为rom而不是利用ip核生成，虽然代码层面方便了不少，但是占用资源过多，编译过程非常的慢，特别是当现在项目比较大，需要多次编译烧录检查，这样的编译时间成本就显得有些得不偿失，在以后还是多利用ip核，而不要在代码中生命容量较大的rom。

九、实验中遇到的问题及解决方法

在一开始尝试的时候每次键盘输入都会导致全屏输出，后来发现是键盘模块和VGA模块共用了一个时钟，导致键盘采集频率过高，一次输入被当作成大量输入，将键盘时钟频率调低后又出现了字符显示不清晰的问题，经过多次调试才找到合适的时序逻辑能够很好的显示字符。

在根据h_addr和v_addr计算出其在ascii[]中的位置和对应的rom_temp的位置时使用了不少乘除法，而fpga计算乘除法很慢，所以有如下修改，该成位运算后提高效率。

```
//计算c,r,x,y 得出字符显示的点阵位置
    assign c=h_addr/9;
    assign r=v_addr>>4;

    assign x=v_addr-(r<<4);
    assign y=h_addr-(c<<3+c);

//原来：
    assign c=h_addr/9;
    assign r=v_addr>>4;
    assign x=v_addr%16;
    assign y=h_addr%9;
```

十、 意见与建议

无