

FPGA 技术培训学习总结与实验心得

——对 VGA 实验电路的改进

姓 名：段 磊

日 期：2007/4/20

目 录

1. 学习感受.....	1
2. 对VGA实验电路的改进.....	2
2.1 VGA接口.....	2
2.2 系统实现.....	3
2.3 VHDL程序设计.....	5
2.4 NIOS程序设计.....	9
2.5 实验结果.....	12
3. 致谢.....	13
4. 参考文献.....	13

1. 学习感受

我在一家通信公司从事研发设计研发工作。几年前的一次偶然机会让我初次接触到 FPGA，没想到过这么普通的一块集成电路居然可以根据需要设计成各种电路。从那以后，我就对 FPGA 产生了极大兴趣。后来，我了解到除了用画原理图的方法设计外，还可以用编程的方法，而且这种方法的可移植性更强，于是我买来参考书开始学习。但是苦于缺少这方面的基础，并且周围也没有精通于此的人能给予足够的指点和帮助，致使我在自学过程中遇到很多困惑，几次陷入停顿。

去年，当我在杂志上看到中电网正在网上组织 FPGA 技术培训时，我感到这个机会太难得了，多年以来渴望全面、系统学习 FPGA 的愿望终于可以实现了，于是我立即报名参加。我过去参加的培训全都是面对面的，学生有问题可以随时向老师提问，但是我对网络教学这种新模式还十分陌生，对学习效果也曾经有过一些担心。后来事实证明我的担心完全是没有必要的。首先，学员不用脱产学习，可以自由安排时间学习，同时免去了上课路途上的劳顿之苦。其次，所有学习内容都被制作成多媒体形式，图文声并茂，可以反复浏览，使得学习记忆深刻，方便复习。而最重要的是，我们能够通过电子邮件的方式直接向著名的孟宪元教授请教问题，得到一对一的辅导。这样的学习模式真是一举多得，中电网为我们想得很周到。

通过几个月的学习，我学到了许多新知识。比如：FPGA 的发展历程，FPGA 的内部基本结构、FPGA 的设计原则、流程与方法，VHDL 语言的特点和编程技巧，时序仿真与系统优化，嵌入式 NIOS 内核的开发，FPGA 在诸多领域的应用等。所有这些都极大的拓宽了我知识面。为了学好 FPGA，我反复做实验，通过实验使我对 FPGA 和 VHDL 语言有了更深入的认识和了解。一个学期下来，我现在已经可以独立设计一些相对复杂的电路，如 4*4 键盘控制电路、8 位并行总线与 SPI 总线转换电路等，而且还改进了实验教材中的 VGA 显示电路。如果没有中电网的老师们和孟宪元教授的辛勤付出，我是取得不了的这些进步的。

由于一些技术问题暂时还没有被解决，再加上我的本职工作也很忙，所以我目前还无法提交一份完整的技术论文。因此，请允许我在下面内容中将已有的研究成果以实验心得的方式向各评审老师汇报，希望你们能多谅解，谢谢。

2. 对 VGA 实验电路的改进

在 THCI-1 实验套件的《实验教材》中，有多个实验都是以 VGA 显示器作为人机接口来输出系统状态或计算结果的。与 7 段数码管和 LED 这些输出方式相比，VGA 显示器可以显示更多的文字或图形信息。与串口比起来，由 FPGA 直接输出 VGA 信号可以省去体积庞大的 PC 主机，因此应用起来十分方便。

《实验教材》中的 VGA 电路将 VGA 显示器模拟成一个 4 位 7 段数码管。这样在方便了实验的同时也降低了 VGA 显示器在文字和图形显示方面的优势。因此我考虑在此基础上改进电路，充分发挥 VGA 显示器的特点，并使之与 NIOS 顺利对接，这样就能够按照程序员的意愿，在屏幕上显示任意的文字或图形。

2.1 VGA 接口

最初的 PC 显示标准（包括 MDA, CGA, EGA, VGA, XGA 等）是由 IBM 制定的。视频电子标准协会（VESA）为了进一步统一显示器的视频显示标准，对以前的标准进行了修订和说明，并同时又制定了一些超过 VGA, XGA 的标准。各显示器和显卡的生产厂家为了保持最大的产品兼容性，对这两家的标准都予以支持。这里以 VESA 的标准为例对 VGA 接口进行说明。

VGA 接口为显示器提供两类信号，一类是数据信号，一类是控制信号。数据信号包括红（Red）、绿（Green）、蓝（Blue）信号，简称 RGB 信号。控制信号包括水平同步信号和垂直同步信号。向显示器输出不同的分辨率时，水平同步信号和垂直同步信号的频率也不同。下表是部分显示模式所使用的频率。

分辨率	垂直同步信号频率 Hz	水平同步信号频率 kHz	图像时钟频率 MHz	同步极性 H/V
640*480	75.000	37.500	31.500	-/-
800*600	75.000	46.875	49.500	+/+
800*600	72.188	48.077	50.000	+/+
1024*768	60.004	48.363	65.000	-/-
1024*768	70.069	56.476	75.000	-/-

从表中可以看出，当分辨率为 640*480 时，垂直同步信号是 75Hz，水平同步信号是 37.5kHz，图像时钟是 31.5MHz，垂直同步信号和水平同步信号都采用低电平同步方式。这些频率之间存在以下关系：

$$31.5\text{MHz} / 37.5\text{kHz} = 840 \quad \dots\dots\dots (1)$$

$$840 - 640 = 200 \quad \dots\dots\dots (2)$$

$$37.5\text{kHz} / 75\text{Hz} = 500 \quad \dots\dots\dots (3)$$

$$500 - 480 = 20 \quad \dots\dots\dots (4)$$

式 (1)、(2) 说明分辨率为 640*480 时，显示器每行占用 840 个数据位，其中 640 个数据位用于显示像素，另外 200 个数据位用于输出水平同步信号和水平消隐信号。式 (3)、(4) 则说明垂直方向有 500 行，其中 480 行用于显示像素，其余 20 行用于输出垂直同步信号和垂直消隐信号。可见，如果想在屏幕的某个位置显示一个某种颜色的点，只需在该点所对应的那个数据位输出对应的 RGB 信号即可。

水平同步信号和垂直同步信号都采用低电平同步方式。通过示波器观察发现水平同步信号低电平约保持 2us，相当于 63 个数据位长度。垂直同步信号低电平约保持 80us，相当于 3 行数据位长度。

2.2 系统实现

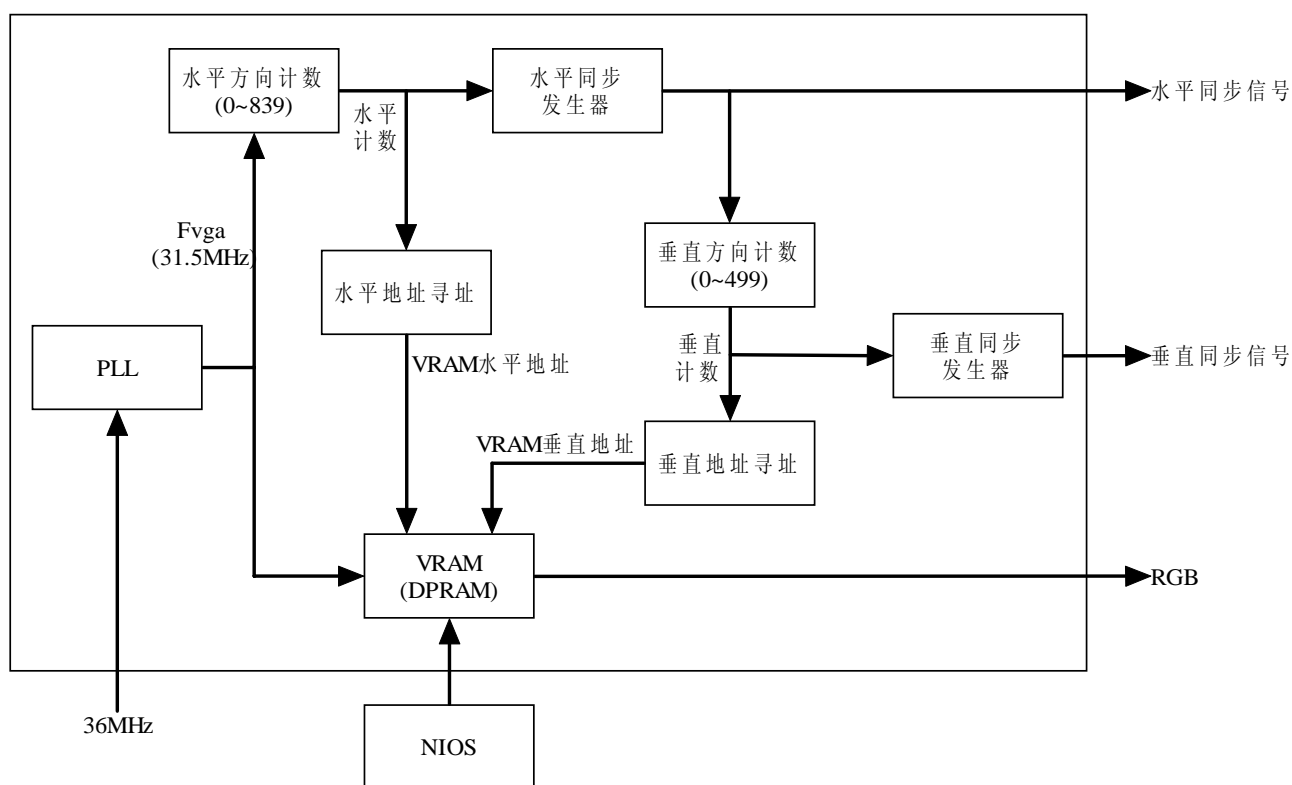
在了解了 VGA 的显示原理之后，我们就会发现，输出标准 VGA 信号的关键是要产生正确的图像时钟频率。有了这个时钟信号，就可以通过计数分频的方法在规定的时刻输出 RGB 信号、垂直同步信号和水平同步信号。

此次改进的 VGA 实验电路的输出分辨率设定在 640*480，需要为其提供 31.5MHz 的时钟信号。THCII-1 实验板上晶振的频率是 50MHz，要想通过 FPGA 内部的锁相环 (PLL) 将时钟调整到 31.5MHz 不太容易，因此将晶振更换为 36MHz，这样经过 7 倍频，8 分频后就可获得 31.5MHz 时钟。

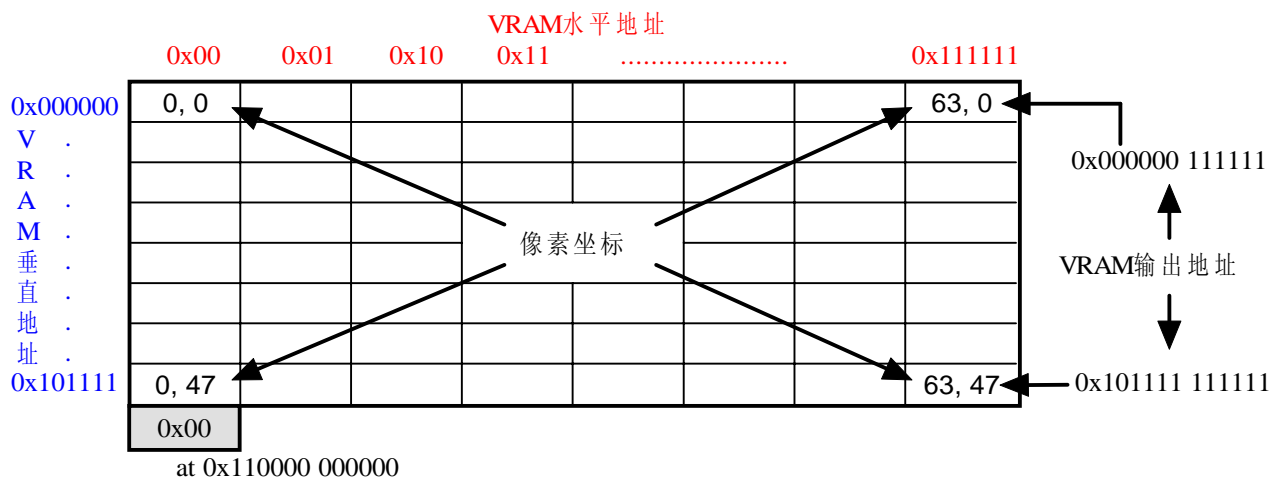
接下来的问题是 NIOS 如何将数据发送给 VGA 显示电路呢？这里参考了 PC 机中显卡的工作方式。即在 NIOS 和 VGA 显示电路之间增设一个双口 RAM (DPRAM) 作为显示存储器 (VRAM)。NIOS 每次把要显示的内容写入 VRAM 中，VGA 显示电路根据显示时序依次读

出 VRAM 中的每个字节。因为每个像素在 VRAM 中都有对应的地址，所以 NIOS 只要修改 VRAM 的内容就可以实现控制显示输出的目的。当向某字节写入 0x00 时，RGB 输出 0 电平，对应的像素显示黑色；当写入 0xFF 时，则 RGB 输出最高电平，对应的像素显示白色。

THCII-1 实验板的VGA输出颜色数为 $256 (=2^8)$ 色，正好可以用VRAM的一个字节表示一个像素。全屏幕共有 $307200 (=640*480)$ 个像素，需要 300k字节的VRAM。这么大的VRAM 在实验板上很难用FPGA (EP2C5) 单独来实现。如果用屏幕上 $10*10$ 个像素所组成的矩形面积表示一个逻辑像素，则人感受到的分辨率就变为 $64*48$ ，此时VRAM仅占 3072 个字节，这在EP2C5 内部实现起来就容易多了。此外，还应再多增加一个值永远为 0x00 的字节。在VGA 电路输出同步信号和消隐信号时，输出地址都指向这个特数字节，使RGB输出信号为 0 电平。按照以上的设计要求，VRAM地址范围从 0x000 至 0xC00，共占用 12 条地址线。



VGA 显示电路原理框图



VRAM 地址表

2.3 VHDL 程序设计

1) 实体部分——实体部分定义了 VGA 显示电路的所有接口。

```
entity VGA is
  Port
  (
    Fosc      : in std_logic;           -- 36MHz 时钟信号。
    Data      : in std_logic_vector(7 downto 0); --VRAM 数据线
    Address   : in std_logic_vector(11 downto 0); --VRAM 写地址
    CE       : in std_logic;           --片选信号，高电平有效
    WREn     : in std_logic;           --写允许信号，高电平有效
    HS       : out std_logic;           --水平同步信号
    VS       : out std_logic;           --垂直同步信号
    RGB      : out std_logic_vector(7 downto 0) --色彩信号
  );
end entity;
```

2) 常量和信号

```
constant HSMax      : integer := 840;           --每行 840 个数据位
constant HSNegative : integer := 63;           --水平同步信号低电平占 63 个数据位
constant VSMax     : integer := 500;           --每帧有 500 行
constant VSNegative : integer := 3;           --垂直同步信号低电平占 3 行数据位

constant XMax      : integer := 640;           --水平 640 个像素
constant YMax     : integer := 480;           --垂直 480 个像素

constant Left     : integer := 184;           --每行首像素位置
constant Right    : integer := Left + XMax - 1; --每行末像素位置
constant Top      : integer := 16;           --首行位置
```

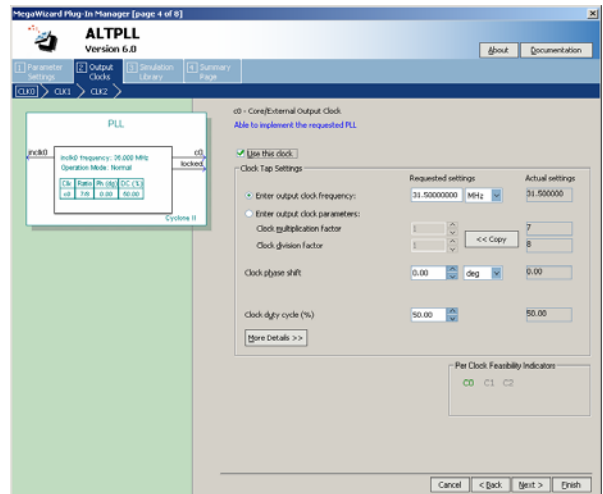
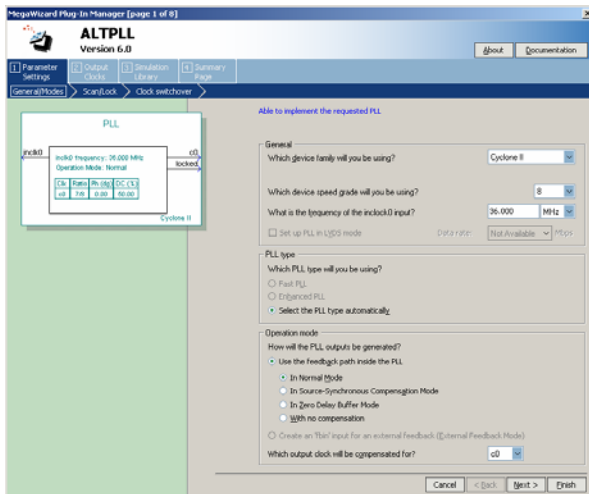
```

constant Bottom      : integer := Top + (YMax - 1); --末行位置

constant PixelWide   : integer := XMax / 64;      --逻辑像素宽度 (10pixel)
constant PixelHigh   : integer := YMax / 48;      --逻辑像素高度 (10pixel)

signal Fvga          : std_logic;                --31.5MHz 时钟信号
signal H              : std_logic;                --垂直同步计数允许信号
signal XAddress       : std_logic_vector(11 downto 0); --VRAM 水平地址
signal YAddress       : std_logic_vector(5 downto 0); --VRAM 垂直地址
signal VRA            : std_logic_vector(11 downto 0); --VRAM 输出地址
    
```

3) PLL 模块对 36MHz 时钟进行 7 倍频和 8 分频，得到 31.5MHz 时钟。



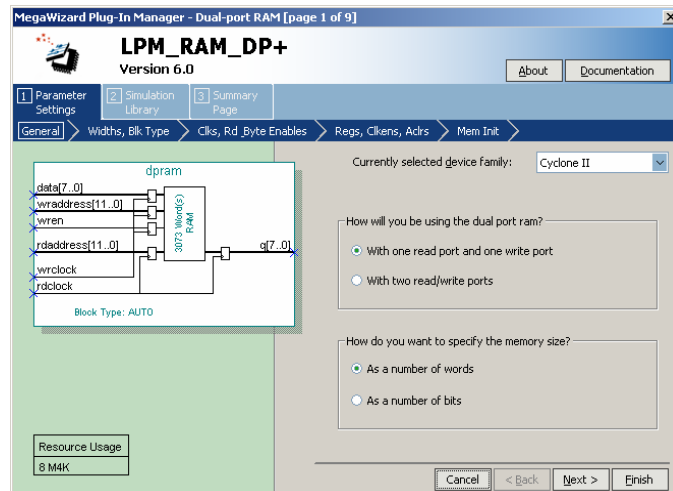
Modular_0:PLL

port map

```

(
    inclk0 => Fosc,      --输入 36MHz 时钟
    c0 => Fvga           --输出 31.5MHz 时钟
);
    
```

4) DPRAM 模块作为 VRAM，为每个像素提供映射地址。



```
Modular : DPRAM
```

```
port map
```

```
(
```

```
    data      => Data,          --VRAM 写入数据, 由 NIOS 写入
    rdaddress => VRA,           --VRAM 输出地址, VGA 显示电路提供
    rdclock   => Fvga,         --VRAM 输出时钟, 31.5MHz
    wraddress => Address,       --VRAM 写入地址
    wrclock   => Fosc,         --VRAM 写入时钟
    wren      => WREn and CE,   --VRAM 写允许信号, WREn 和 CE 为 1 时有效
    q        => RGB            --RGB 输出信号
```

```
);
```

5) 水平同步信号的产生和 VRAM 水平地址寻址

```
process(Fvga)
```

```
variable HSValue : integer range 0 to HSMax - 1 := 0;
```

```
begin
```

```
    if rising_edge(Fvga) then
```

```
--在 Fvga (31.5MHz) 的上升沿循环计数 (0-839)
```

```
        if HSValue = HSMax - 1 then
```

```
            HSValue := 0;
```

```
        else
```

```
            HSValue := HSValue + 1;
```

```
        end if;
```

```
--在计数值小于 63 时 (0 至 62), 水平同步信号输出低电平, 否则输出高电平
```

```
        if HSValue < HSNegative then
```

```
            HS <= '0';
```

```
        else
```

```
            HS <= '1';
```

```
        end if;
```

```
--在计数值等于 839 时, 垂直同步计数允许信号为 1, 否则为 0
```

```
        if HSValue = HSMax - 1 then
```

```
            H <= '1';
```

```
        else
```

```
            H <= '0';
```

```
        end if;
```

```
--每行从首像素到末像素, 每 10 个像素对应一个 VRAM 水平地址, 共 64 个水平地址
```

```
if HSValue >= Left + 0 * PixelWide and HSValue < Left + 1 * PixelWide then XAddress <= "000000000000";
```

```
elsif HSValue >= Left + 1 * PixelWide and HSValue < Left + 2 * PixelWide then XAddress <= "000000000001";
```

```
elsif HSValue >= Left + 2 * PixelWide and HSValue < Left + 3 * PixelWide then XAddress <= "000000000010";
```

```
elsif HSValue >= Left + 3 * PixelWide and HSValue < Left + 4 * PixelWide then XAddress <= "000000000011";
```

```
.....--中间雷同, 省略
```

```
elsif HSValue >= Left + 63 * PixelWide and HSValue < Left + 64 * PixelWide then XAddress <=
```

```

"000000111111";
--超出首像素和末像素时，输出默认地址 0x110000000000 的 0x00
else XAddress <= "110000000000";
    end if;
    end if;
end process;

```

6) 垂直同步信号的产生和 VRAM 垂直地址寻址

```

process(Fvga, H)
variable VSValue : integer range 0 to VSMax - 1 := 0;
begin
    if rising_edge(Fvga) then
--在 Fvga (31.5MHz) 的上升沿，当垂直同步计数允许信号为 1 时，对行数循环计数 (0~479)
        if H = '1' then
            if VSValue = VSMax - 1 then
                VSValue := 0;
            else
                VSValue := VSValue + 1;
            end if;
        else
            VSValue := VSValue;
        end if;
--当行计数小于 3 时，垂直同步信号输出低电平，否则输出高电平
        if VSValue < VSNegative then
            VS <= '0';
        else
            VS <= '1';
        end if;
--从首行到末行，每 10 个行对应一个 VRAM 垂直地址，共 48 个水平地址
        if VSValue >= Top + 0 * PixelHigh and VSValue < Top + 1 * PixelHigh then YAddress <= "000000";
        elsif VSValue >= Top + 1 * PixelHigh and VSValue < Top + 2 * PixelHigh then YAddress <= "000001";
        elsif VSValue >= Top + 2 * PixelHigh and VSValue < Top + 3 * PixelHigh then YAddress <= "000010";
        elsif VSValue >= Top + 3 * PixelHigh and VSValue < Top + 4 * PixelHigh then YAddress <= "000011";

        .....--中间雷同，省略

        elsif VSValue >= Top + 47 * PixelHigh and VSValue < Top + 48 * PixelHigh then YAddress <= "101111";
--超出首行和末行时，输出默认地址 0x110000000000 的垂直地址部分 0x110000
        else YAddress <= "110000";
            end if;
        end if;
    end process;

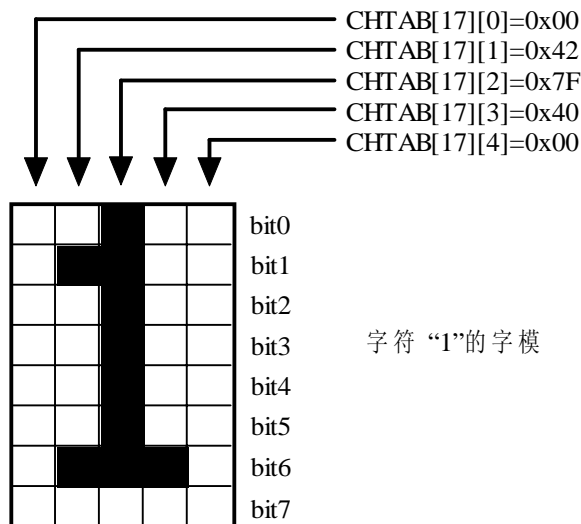
```

7) VRAM 输出地址的合成

```
VRA <= YAddress & "000000" or XAddress;
```

2.4 NIOS 程序设计

- 1) 字模表。每个字符均由 5*7 个逻辑像素组成，占用 5 个字节。字符“1”的字模如下图所示。任意字符的 ASCII 码经过差值计算后就能得到该字符在字模表中的位置。



字符“1”的字模

```
unsigned char CHTAB[][5]=
{
  {0x00, 0x00, 0x00, 0x00, 0x00},    //" "=00h
  {0x00, 0x00, 0x4F, 0x00, 0x00},    //"!"=01h
  {0x00, 0x07, 0x00, 0x07, 0x00},    //""=02h
  {0x14, 0x7F, 0x14, 0x7F, 0x14},    //"# "=03h
  {0x24, 0x2A, 0x7F, 0x2A, 0x12},    //"$ "=04h
  {0x23, 0x13, 0x08, 0x64, 0x62},    //"% "=05h
  {0x36, 0x49, 0x55, 0x22, 0x50},    //"& "=06h
  {0x00, 0x05, 0x03, 0x00, 0x00},    //""=07h
  {0x00, 0x1C, 0x22, 0x41, 0x00},    //"( "=08h
  {0x00, 0x41, 0x22, 0x1C, 0x00},    //") "=09h
  {0x14, 0x08, 0x3E, 0x08, 0x14},    //"* "=0Ah
  {0x08, 0x08, 0x3E, 0x08, 0x08},    //"+ "=0Bh
  {0x00, 0x50, 0x30, 0x00, 0x00},    //", "=0Ch
  {0x08, 0x08, 0x08, 0x08, 0x08},    //"- "=0Dh
  {0x00, 0x60, 0x60, 0x00, 0x00},    //". "=0Eh
  {0x20, 0x10, 0x08, 0x04, 0x02},    //"/ "=0Fh
  {0x3E, 0x51, 0x49, 0x45, 0x3E},    //"0 "=10h
  {0x00, 0x42, 0x7F, 0x40, 0x00},    //"1 "=11h
  {0x42, 0x61, 0x51, 0x49, 0x46},    //"2 "=12h
  {0x21, 0x41, 0x45, 0x4B, 0x31},    //"3 "=13h
  {0x18, 0x14, 0x12, 0x7F, 0x10},    //"4 "=14h
  {0x27, 0x45, 0x45, 0x45, 0x39},    //"5 "=15h
```

```
{0x3C, 0x4A, 0x49, 0x49, 0x30},    ///<6"=16h
{0x01, 0x01, 0x79, 0x05, 0x03},    ///<7"=17h
{0x36, 0x49, 0x49, 0x49, 0x36},    ///<8"=18h
{0x06, 0x49, 0x49, 0x29, 0x1E},    ///<9"=19h
{0x00, 0x36, 0x36, 0x00, 0x00},    ///<:"=1Ah
{0x00, 0x56, 0x36, 0x00, 0x00},    ///<;"=1Bh
{0x08, 0x14, 0x22, 0x41, 0x00},    ///<<"=1Ch
{0x14, 0x14, 0x14, 0x14, 0x14},    ///<="=1Dh
{0x00, 0x41, 0x22, 0x14, 0x08},    ///<>"=1Eh
{0x02, 0x01, 0x51, 0x09, 0x06},    ///"=1Fh
{0x32, 0x49, 0x79, 0x41, 0x3E},    ///<@"=20h
{0x7E, 0x11, 0x11, 0x11, 0x7E},    ///<A"=21h
{0x41, 0x7F, 0x49, 0x49, 0x36},    ///<B"=22h
{0x3E, 0x41, 0x41, 0x41, 0x22},    ///<C"=23h
{0x41, 0x7F, 0x41, 0x41, 0x3E},    ///<D"=24h
{0x7F, 0x49, 0x49, 0x49, 0x49},    ///<E"=25h
{0x7F, 0x09, 0x09, 0x09, 0x01},    ///<F"=26h
{0x3E, 0x41, 0x41, 0x49, 0x7A},    ///<G"=27h
{0x7F, 0x08, 0x08, 0x08, 0x7F},    ///<H"=28h
{0x00, 0x41, 0x7F, 0x41, 0x00},    ///<I"=29h
{0x20, 0x40, 0x41, 0x3F, 0x01},    ///<J"=2Ah
{0x7F, 0x08, 0x14, 0x22, 0x41},    ///<K"=2Bh
{0x7F, 0x40, 0x40, 0x40, 0x40},    ///<L"=2Ch
{0x7F, 0x02, 0x0C, 0x02, 0x7F},    ///<M"=2Dh
{0x7F, 0x06, 0x08, 0x30, 0x7F},    ///<N"=2Eh
{0x3E, 0x41, 0x41, 0x41, 0x3E},    ///<O"=2Fh
{0x7F, 0x09, 0x09, 0x09, 0x06},    ///<P"=30h
{0x3E, 0x41, 0x51, 0x21, 0x5E},    ///<Q"=31h
{0x7F, 0x09, 0x19, 0x29, 0x46},    ///<R"=32h
{0x26, 0x49, 0x49, 0x49, 0x32},    ///<S"=33h
{0x01, 0x01, 0x7F, 0x01, 0x01},    ///<T"=34h
{0x3F, 0x40, 0x40, 0x40, 0x3F},    ///<U"=35h
{0x1F, 0x20, 0x40, 0x20, 0x1F},    ///<V"=36h
{0x7F, 0x20, 0x18, 0x20, 0x7F},    ///<W"=37h
{0x63, 0x14, 0x08, 0x14, 0x63},    ///<X"=38h
{0x07, 0x08, 0x70, 0x08, 0x07},    ///<Y"=39h
{0x61, 0x51, 0x49, 0x45, 0x43},    ///<Z"=3Ah
{0x00, 0x7F, 0x41, 0x41, 0x00},    ///<["=3Bh
{0x02, 0x04, 0x08, 0x10, 0x20},    ///<\"=3Ch
{0x00, 0x41, 0x41, 0x7F, 0x00},    ///<]"=3Dh
{0x04, 0x02, 0x01, 0x02, 0x04},    ///<^"=3Eh
{0x40, 0x40, 0x40, 0x40, 0x40},    ///<_"=3Fh
{0x01, 0x02, 0x04, 0x00, 0x00},    ///<`"=40h
{0x20, 0x54, 0x54, 0x54, 0x78},    ///<a"=41h</pre
```

```

{0x7F, 0x48, 0x44, 0x44, 0x38}, // "b"=42h
{0x38, 0x44, 0x44, 0x44, 0x28}, // "c"=43h
{0x38, 0x44, 0x44, 0x48, 0x7F}, // "d"=44h
{0x38, 0x54, 0x54, 0x54, 0x18}, // "e"=45h
{0x00, 0x08, 0x7E, 0x09, 0x02}, // "f"=46h
{0x0C, 0x52, 0x52, 0x4C, 0x3E}, // "g"=47h
{0x7F, 0x08, 0x04, 0x04, 0x78}, // "h"=48h
{0x00, 0x44, 0x7D, 0x40, 0x00}, // "i"=49h
{0x20, 0x40, 0x44, 0x3D, 0x00}, // "j"=4Ah
{0x00, 0x7F, 0x10, 0x28, 0x44}, // "k"=4Bh
{0x00, 0x41, 0x7F, 0x40, 0x00}, // "l"=4Ch
{0x7C, 0x04, 0x78, 0x04, 0x78}, // "m"=4Dh
{0x7C, 0x08, 0x04, 0x04, 0x78}, // "n"=4Eh
{0x38, 0x44, 0x44, 0x44, 0x38}, // "o"=4Fh
{0x7E, 0x0C, 0x12, 0x12, 0x0C}, // "p"=50h
{0x0C, 0x12, 0x12, 0x0C, 0x7E}, // "q"=51h
{0x7C, 0x08, 0x04, 0x04, 0x08}, // "r"=52h
{0x58, 0x54, 0x54, 0x54, 0x64}, // "s"=53h
{0x04, 0x3F, 0x44, 0x40, 0x20}, // "t"=54h
{0x3C, 0x40, 0x40, 0x3C, 0x40}, // "u"=55h
{0x1C, 0x20, 0x40, 0x20, 0x1C}, // "v"=56h
{0x3C, 0x40, 0x30, 0x40, 0x3C}, // "w"=57h
{0x44, 0x28, 0x10, 0x28, 0x44}, // "x"=58h
{0x1C, 0xA0, 0xA0, 0x90, 0x7C}, // "y"=59h
{0x44, 0x64, 0x54, 0x4C, 0x44}, // "z"=5Ah
{0x00, 0x08, 0x36, 0x41, 0x00}, // "{"=5Bh
{0x00, 0x00, 0x77, 0x00, 0x00}, // "|"=5Ch
{0x00, 0x41, 0x36, 0x08, 0x00}, // "}"=5Dh
{0x02, 0x01, 0x02, 0x04, 0x02}, // "~"=5Fh
{0x7F, 0x7F, 0x7F, 0x7F, 0x7F} // " "=60h
};

```

- 2) 显示像素函数，负责向 VRAM 中指定行和列上的字节写入颜色值。VGA_0_BASE 为 VRAM 的首地址。

```

void Pixel(unsigned char row, unsigned char column, unsigned char color)
{
    unsigned short offset;
    offset = row * 64 + column;
    IOWR(VGA_0_BASE, offset, color);
}

```

- 3) 清屏函数，负责向 VRAM 中的所有字节写 0x00，即全屏显示黑色。

```

void ClearScreen(void)
{

```

```
unsigned short p;  
for(p = 0; p < 3072; p ++)  
    IOWR(VGA_0_BASE, p, 0);  
}
```

4) 字符显示函数

```
void Character(unsigned char row, unsigned char column, unsigned char *CH, unsigned char  
frontcolor, unsigned char backcolor)  
{  
    unsigned char i, h, color;  
    for(h = 0; h < 5; h ++)  
        //遍历字模占用的 5 个字节  
        {  
            for(i = 0; i < 8; i ++)  
                //遍历字模中每个字节的 8 个 bit  
                {  
                    if(1 == ((*CH + h) >> i) & 1)  
                        {  
                            color = frontcolor;           //如果某 bit 为 1, 则显示前景颜色  
                        }  
                    else  
                        {  
                            color = backcolor;           //如果某 bit 为 0, 则显示背景颜色值  
                        }  
                    Pixel(row + i, column + h, color);    //显示像素  
                }  
            }  
        }  
}
```

2.5 实验结果

经过实验证明, 本人对 VGA 实验电路所做的改进设计完全符合当初的设想。如果在 Flash 中加入汉字库, 还可以显示汉字。



3. 致谢

最后，我要向中电网表示衷心的感谢，谢谢你们为中国的广大电子技术人员提供了一个这么好的学习技术的平台和继续深造的机会。感谢中电网的所有领导、老师和工作人员提供的热情帮助与周到服务。感谢孟宪元教授的授课，感谢他在百忙之中还为我们提供技术辅导。感谢 Altera 公司、骏龙科技有限公司、文晔科技股份有限公司的技术服务人员提供的无偿技术支持。

4. 参考文献

- 1) 戴梅萼等. 微型计算机技术及应用. 北京: 清华大学出版社, 1996
- 2) 赵俊超等. 集成电路设计 VHDL 教程. 北京: 北京希望电子出版社, 2002