一、
首先软件实现画点函数

二、
将图像区的部分地址分配给光标，用于光标坐标的写入
光标坐标的数据利用一个周期的 HWDATA 传递，其中 0-6 位为 x 坐标，7-11 位为 y 坐标

```
//地址选择写文字数据还是图像数据
assign sel_console = (last_HADDR[23:0]== 0);
assign sel_image = (last_HADDR[22:0] != 23'b000_0000_0000_0000_0000_0000) && (last_HADDR[23] != 1'b1);

always @(posedge HCLK or negedge HRESETn)
begin
  if(!HRESETn)
  begin
      r_cursor_x <= 7'b000_0000;
      r_cursor_y <= 5'b0_0000;
  end
  else if (last_HWRITE & last_HSEL & last_HTRANS[1] & HREADYOUT & (last_HADDR==32'h5080_0000) )
  begin
      r_cursor_x <= HWDATA[6:0];
      r_cursor_y <= HWDATA[11:7];
  end
end
```

并在 vga_console 模块里，将对应地址的数据写入 cur_next 寄存器，即可控制字符写入的坐标，同时考虑字符串的显示，保留 cur_next 原本的逻辑

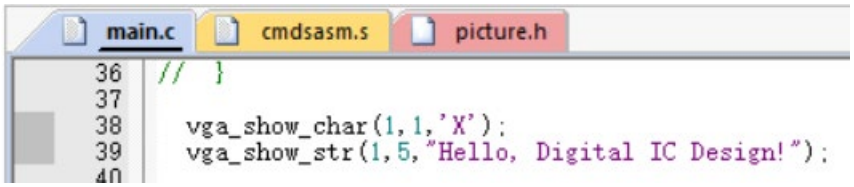
```
//Next Cursor Position logic
always @*
begin
  if (last_address == 32'h5080_0000)
    begin
      cur_x_next = cursor_x;
      cur_y_next = cursor_y;
    end
  else
    begin
      cur_x_next = (new_line) ? 2 :
                   (back_space && cur_x_reg) ? cur_x_reg - 1 :
                   (font_we && ~back_space && ~scroll) ? cur_x_reg + 1 : cur_x_reg;

      cur_y_next = (cur_y_reg == MAX_Y-1) ? cur_y_reg :
                   ((new_line) ? cur_y_reg + 1 : cur_y_reg );
    end

end
```

编写 C 语言函数并封装，此时地址数据的转换与之前的设定相对应（0-6 位为 x 坐标，7-11 位为 y 坐标）

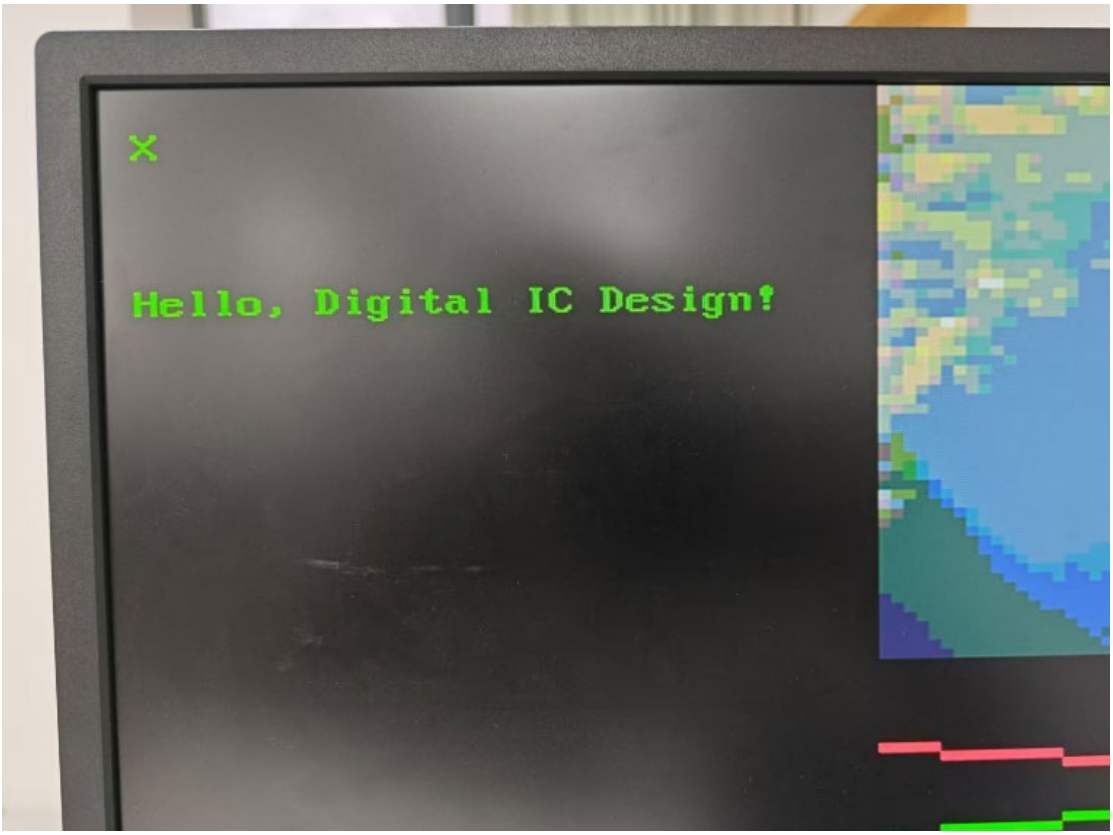
```c
void vga_show_char(volatile int x, volatile int y, char ch)
{
    // 将x和y坐标转换成VGA_CUR_REG的值
    volatile unsigned int address = (y << 7) | x;

    // 设置VGA字符地址
    VGA_CUR_REG = address;

    // 在指定位置写入字符
    VGA_TEXT_REG = ch;
}
```

```c
void vga_show_str(volatile int x, volatile int y, const char* str)
{
    volatile int current_x = x;
    volatile int current_y = y;

    while (*str) {
        // 计算字符地址
        unsigned int address = (current_y << 7) | current_x;
        VGA_CUR_REG = address;      // 设置字符地址
        VGA_TEXT_REG = *str++;      // 写入字符并移到下一个字符

        // 更新坐标
        current_x++;
        if (current_x >= VGA_TEXT_WIDTH) {  // 到行末时换行
            current_x = 0;
            current_y++;
        }
    }
}
```

综合烧录测试



经过测试，能在自定义坐标位置下，显示所输入的字符


X
Hello, Digital IC Design!

三、

在 VGASYS 模块中为字体颜色分配地址 0x5080_0004

```
assign sel_coruse = (last_HADDR == 32'h5080_0000);
assign sel_text_color = (last_HADDR==32'h5080_0004);
```

```
//Set text color and background color
always @(posedge HCLK or negedge HRESETn)
begin
  if(!HRESETn)
  begin
      r_text_color <= 8'b0000_0000;
      r_text_back_color <= 8'b0000_0000;
  end
  else if (last_HWRITE & last_HSEL & last_HTRANS[1] & HREADYOUT & sel_text_color )
  begin
    r_text_color <= HWDATA[7:0];
    r_text_back_color <= HWDATA[15:8];
  end
end
```

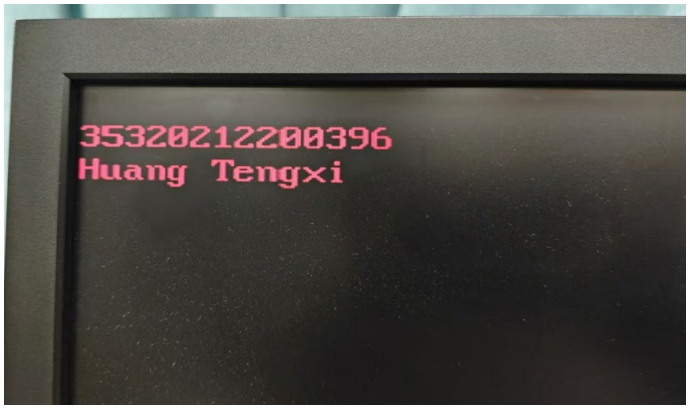
在 vga_console 中添加相关寄存器

```
//Color Generation
always @*
begin
    begin
      font_rgb = (font_bit) ? text_color : text_back_color;
      font_inv_rgb = (font_bit) ? text_back_color : text_color;
    end
```

综合和在 kiel 中编写相应的程序驱动显示

```
#define VGA_CUR_REG          *((volatile int*)0x50800000)
#define VGA_TEXT_COLOR_REG   *((volatile int*)0x50800004)

  VGA_TEXT_COLOR_REG = LIGHT_RED;
  vga_show_str(0,1,"353202122003 96");
  vga_show_str(0,2,"Huang Tengxi");
```

显示效果如下，能改变所打印字符的颜色



四、
阅读代码后，发现代码中有包含文本区宽度的参数，为这个参数分配一个寄存器和地址

```
//Select the rgb color for a particular region
always @*
begin
  if(!HRESETn)
    cin <= 8'h00;
  else
    if(pixel_x[9:0]< r_text_width
      cin <= console_rgb ;
    else
      cin <= image_rgb;
end
```

```
assign addr_w = address_reg[15:0];
assign din = image_data;

assign img_x = pixel_x[9:0]-text_width;
assign img_y = pixel_y[9:0];
```

分配的地址为 5080_0008

```
assign sel_text_width = (last_HADDR==32'h5080_0008);

//Set text region width
always @(posedge HCLK or negedge HRESETn)
begin
  if(!HRESETn)
    r_text_width <= 9'b0_1111_0000;
  else if (last_HWRITE & last_HSEL & last_HTRANS[1] & HREADYOUT & sel_text_width )
    r_text_width <= HWDATA[9:0];
end
```

同时，在原本的图像区中，显示的分辨率最大为 128*128，如果文本区宽度小于 240，那么多出的图像区将显示错误的信息，因此需要为图像区显示地址的 x 地位宽从原来的 7 位增加至 8 位，这样才能实现全屏显示图像的要求
将所有与图像相关的地址位宽全增加一位

```
assign addr_w = address_reg[15:0];
assign din = image_data;

assign img_x = pixel_x[9:0]-text_width;
assign img_y = pixel_y[9:0];

assign addr_r = {1'b0, img_y[8:2], img_x[9:2]};
```

双端口 RAM 的地址位宽也需要增加，修改后显示的最大分配率变为 256*128

```
//Frame buffer
dual_port_ram_sync
 #(.ADDR_WIDTH(16), .DATA_WIDTH(8))
 uimage_ram
 ( .clk(clk),
   .reset_n(resetn),
   .we(image_we),
   .addr_a(addr_w),
   .addr_b(addr_r),
   .din_a(din),
   .dout_a(),
   .dout_b(dout)
 );
```

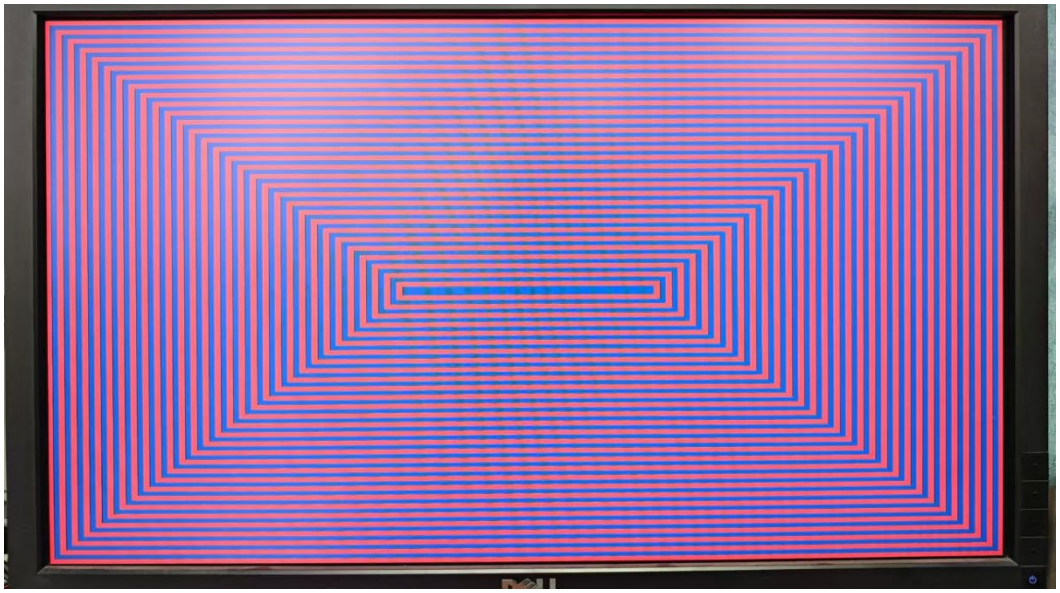
软件中对于地址的换算也需要做对应的调整

```
void vga_point(volatile int x ,volatile int y ,unsigned char color)
{
  PIXEL_ADDRESS= (volatile int*)(VGA_IMAGE_REG + (x<<2) + (y<<10));
  *PIXEL_ADDRESS = color ;
}
```

修改后综合测试，运行如下程序

```
VGA_TEXT_WIDTH_REG = 0;
vga_draw_image(0,0,160,120,IMAGE);
```

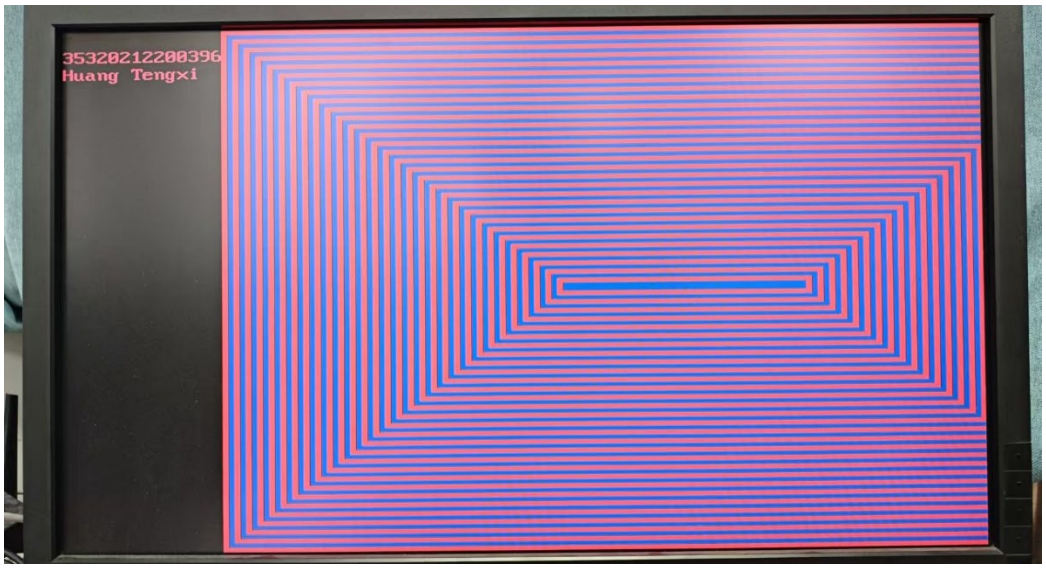
显示效果如下，可见无文本区，全屏正常显示图像



运行如下程序

```
VGA_TEXT_WIDTH_REG = 112;

VGA_TEXT_COLOR_REG = LIGHT_RED;
vga_show_str(0,1,"353202122000396");
vga_show_str(0,2,"Huang Tengxi");

vga_draw_image(0,0,160,120,IMAGE);
```

可以调节文本区宽度



运行如下测试

```
VGA_TEXT_WIDTH_REG = 640;

VGA_TEXT_COLOR_REG = LIGHT_RED;
vga_show_str(0,5,"35320212200396");
vga_show_str(0,6,"Huang Tengxi");

vga_show_char(0,0,'A');
vga_show_char(79,0,'B');
vga_show_char(0,28,'C');
vga_show_char(79,28,'D');
```

可以全屏显示文字

经过以上测试，可见文本区与图像区大小可以自由调节，并均能正常执行程序，显示相应的字符或图像

五、

要实现图像区的高分辨率显示，首先需要为图像数据分配足够大小的 ram，要实现 640*480 的全屏图像显示，需要分配 512*1024*8，即 512kb 大小的 ram

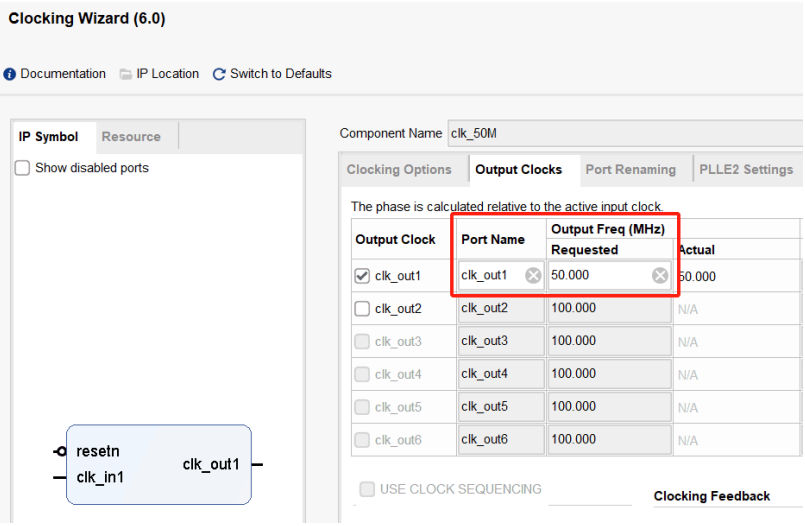
```
//Frame buffer

dual_port_ram_sync
 #(.ADDR_WIDTH(19), .DATA_WIDTH(8))
 uimage_ram
 ( .clk(clk),
   .reset_n(resetn),
   .we(image_we),
   .addr_a(addr_w),
   .addr_b(addr_r),
   .din_a(din),
   .dout_a(),
   .dout_b(dout)
 );
```

```
reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
reg [ADDR_WIDTH-1:0] addr_a_reg;
reg [ADDR_WIDTH-1:0] addr_b_reg;

reg [ADDR_WIDTH:0] reset_addr;
reg [1:0] reset_n_buf;
```

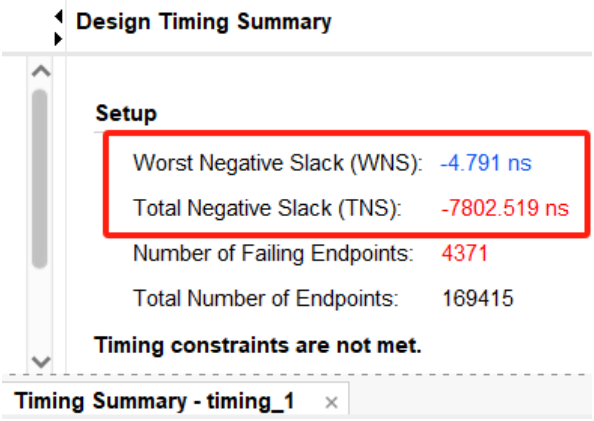
分配对应大小的 ram 后进行综合发现无法进行调试，查看时序报告也未发现异常

使用 DCM IP 核来输出时钟后，可以发现实际的时序问题

```
wire clk_div_vga;
clk_50M u_clk_50M(
.resetn(RESET),
.clk_in1(CLK),
.clk_out1(clk_div_vga)
);

BUFG BUFG_VGA_CLK (
    .O(vgaclk),
    .I(clk_div_vga)
);
```



修改时钟后进行综合，发现布线综合的时序不符合要求，分析发现导致时序问题的原因是 RAM 占据面积过大，导致布线过长，路径延时大于时钟周期（50MHz）



因此需要降低核心频率，在测试后选择以 25MHz 作为 ARM 核的运行频率。同时，为保持 VGA 输出的时序，需要用 50MHz 驱动 VGA 的输出模块

```
wire clk_div;
clk_25M u_clk_25M(
.resetn(RESET),
.clk_in1(vgaclk),
.clk_out1(clk_div)
);

BUFG BUFG_CLK (
    .O(fclk),
    .I(clk_div)
);
```

第一级50M时钟

ARM核心25M时钟

```
VGAInterface uVGAInterface (
    .CLK(VGACLK),
    .COLOUR_IN(cin),
    .cout(rgb),
    .hs(hsync),
    .vs(vsync),
    .addrh(pixel_x),
    .addrv(pixel_y)
    );
```

作出如上修改后，能正常进行综合，现在需要对 vga_image 模块做出对应修改
阅读代码，发现将 img_x 和 img_y 的位数做对应限制即可实现不同分辨率
因此先为 image_zoom 寄存器分配地址

```
//Set image zoom
always @(posedge HCLK or negedge HRESETn)
begin
  if(!HRESETn)
    r_image_zoom <= 3'b000;
  else if (last_HWRITE & last_HSEL & last_HTRANS[1] & HREADYOUT & sel_image_zoom )
    r_image_zoom <= HWDATA[2:0];
end
```

根据 image_zoom 的写入值修改分辨率

```
always @*
begin
  case (image_zoom)

    3'b000:  addr_r <= {1'b0, img_y[8:0], img_x[9:0]};
    3'b001:  addr_r <= {2'b0, img_y[8:1], 1'b0, img_x[9:1]};
    3'b010:  addr_r <= {3'b0, img_y[8:2], 2'b0, img_x[9:2]};

    default:  addr_r <= {1'b0, img_y[8:0], img_x[9:0]};

  endcase

end
```

编写对应软件，运行如下代码后

```
void VGA_INIT(volatile int text_width,
{
  VGA_TEXT_WIDTH_REG = text_width;
  VGA_TEXT_COLOR_REG = text_color;
  VGA_IMAGE_ZOOM_REG = image_zoom;
}

VGA_INIT(0,LIGHT_RED,0);
vga_draw_image(0,0,160,120,IMAGE);
vga_draw_image(319,239,160,120,IMAGE);
```

可以在显示屏上看到如下图像，可以看到图像以 640*480 的分辨率正常显示，全屏均可作为显示区域

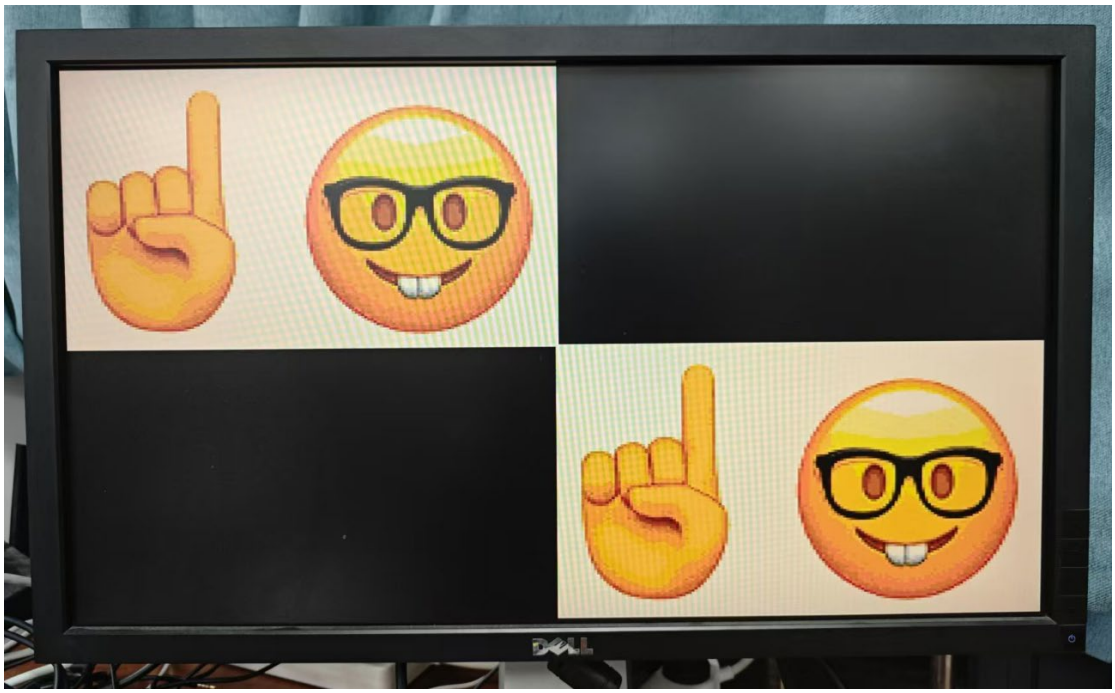
需要注意的是因为 rom 的地址位数增加，之前的画点函数的参数也需要作出对应修改，又因为在编写 Verilog 时将 img_x 和 img_y 的位数统一，因此软件的画点函数修改后可以通用

```
void vga_point(volatile int x ,volatile int y ,unsigned char color)
{

  PIXEL_ADDRESS = (volatile int*)(VGA_IMAGE_REG + (x << 2) + (y << 12));

  *PIXEL_ADDRESS = color ;
}
```

调节分辨率运行以下代码即可实现

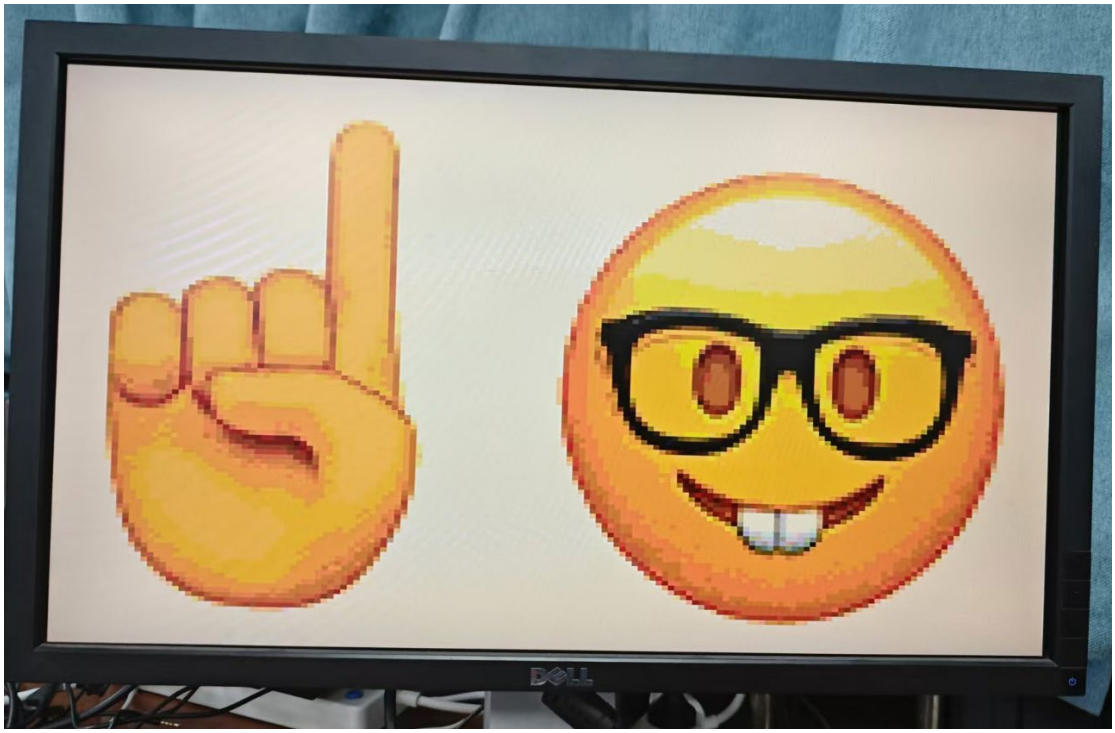
2 倍分辨率显示效果如下

```
VGA_INIT(0,LIGHT_RED,1);
vga_draw_image(0,0,160,120,IMAGE);
vga_draw_image(159,119,160,120,IMAGE);
```



四倍分辨率显示效果如下

```
VGA_INIT(0,LIGHT_RED,2);
vga_draw_image(0,0,160,120,IMAGE);
```



修改图像区的寻址方式，将 x 和 y 的顺序交换，图像区 ram 的大小可以降低至 320kb（640*512）

```
always @*
begin
  case (image_zoom)

    3'b000:  addr_r <= {1'b0, img_x[9:0], img_y[8:0]};
    3'b001:  addr_r <= {1'b0, img_x[9:1], 2'b0, img_y[8:1]};
    3'b010:  addr_r <= {2'b0, img_x[9:2], 3'b0, img_y[8:2]};

    default:  addr_r <= {1'b0, img_x[9:0], img_y[8:0]};

  endcase

end
```

```
// reg [DATA_WIDTH-1:0] ram [2**ADDR_WIDTH-1:0];
reg [DATA_WIDTH-1:0] ram [327679:0];
reg [ADDR_WIDTH-1:0] addr_a_reg;
reg [ADDR_WIDTH-1:0] addr_b_reg;
```

软件部分仅需进行如下修改即可

```
void vga_point(volatile int x ,volatile int y ,unsigned char color)
{
  PIXEL_ADDRESS = (volatile int*)(VGA_IMAGE_REG + (y << 2) + (x << 11)); // 修改为 <<11 以避免隔行问题
  *PIXEL_ADDRESS = color ;
}
```

经测试能正常显示

六、

要实现 4096 色首先需要根据数据手册添加管脚约束

名称	原理图标号	FPGA IO PIN
RED	VGA_R0	F5
	VGA_R1	C6
	VGA_R2	C5
	VGA_R3	B7
GREEN	VGA_G0	B6
	VGA_G1	A6
	VGA_G2	A5
	VGA_G3	D8
BLUE	VGA_B0	C7
	VGA_B1	E6
	VGA_B2	E5
	VGA_B3	E7
H-SYNC	VGA_HSYNC	D7
V-SYNC	VGA_VSYNC	C4

```
set_property PACKAGE_PIN D7 [get_ports VGA_HSYNC]
set_property PACKAGE_PIN C4 [get_ports VGA_VSYNC]
set_property PACKAGE_PIN B7 [get_ports {VGA_RED[3]}]
set_property PACKAGE_PIN C5 [get_ports {VGA_RED[2]}]
set_property PACKAGE_PIN C6 [get_ports {VGA_RED[1]}]
set_property PACKAGE_PIN F5 [get_ports {VGA_RED[0]}]
set_property PACKAGE_PIN D8 [get_ports {VGA_GREEN[3]}]
set_property PACKAGE_PIN A5 [get_ports {VGA_GREEN[2]}]
set_property PACKAGE_PIN A6 [get_ports {VGA_GREEN[1]}]
set_property PACKAGE_PIN B6 [get_ports {VGA_GREEN[0]}]
set_property PACKAGE_PIN E7 [get_ports {VGA_BLUE[3]}]
set_property PACKAGE_PIN E5 [get_ports {VGA_BLUE[2]}]
set_property PACKAGE_PIN E6 [get_ports {VGA_BLUE[1]}]
set_property PACKAGE_PIN C7 [get_ports {VGA_BLUE[0]}]
```

添加管脚约束后将所有原本 8 位 RGB 的端口全修改为 12 位 RGB

```
// VGA IO
output wire     [3:0] VGA_RED,
output wire     [3:0] VGA_GREEN,
output wire     [3:0] VGA_BLUE,
```

同时，需要将图像区的数据宽度由 8 位改至 12 位，这样才能正常显示

```
dual_port_ram_sync
 #(.ADDR_WIDTH(16), .DATA_WIDTH(12))
 uimage_ram
 ( .clk(clk),
   .reset_n(resetn),
   .we(image_we),
   .addr_a(addr_w),
   .addr_b(addr_r),
   .din_a(din),
   .dout_a(),
   .dout_b(dout)
 );
```

经过测试，能正常显示 4096 色，以下为 4096 色全色显示效果