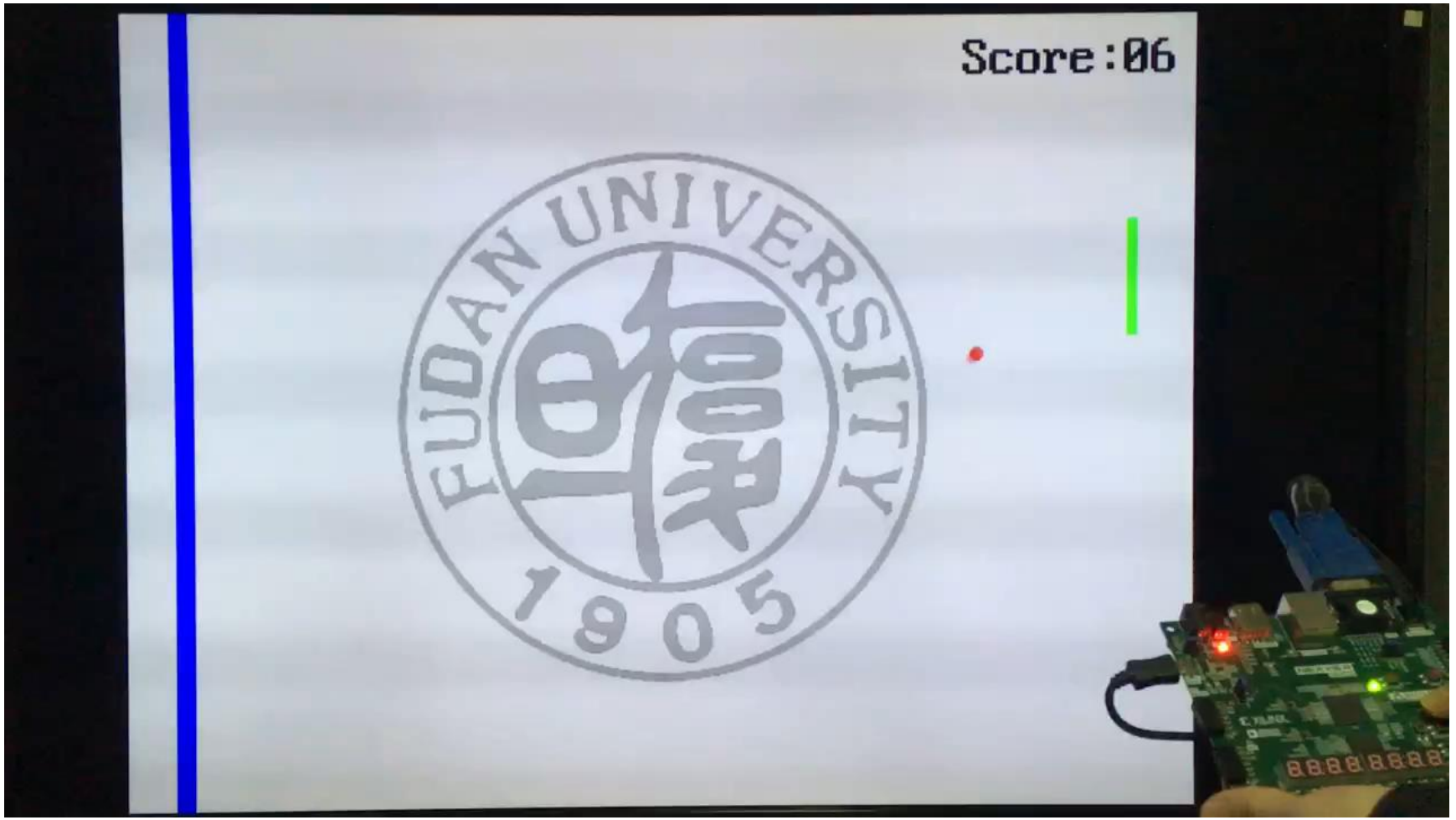
基于FPGA原型的游戏设计

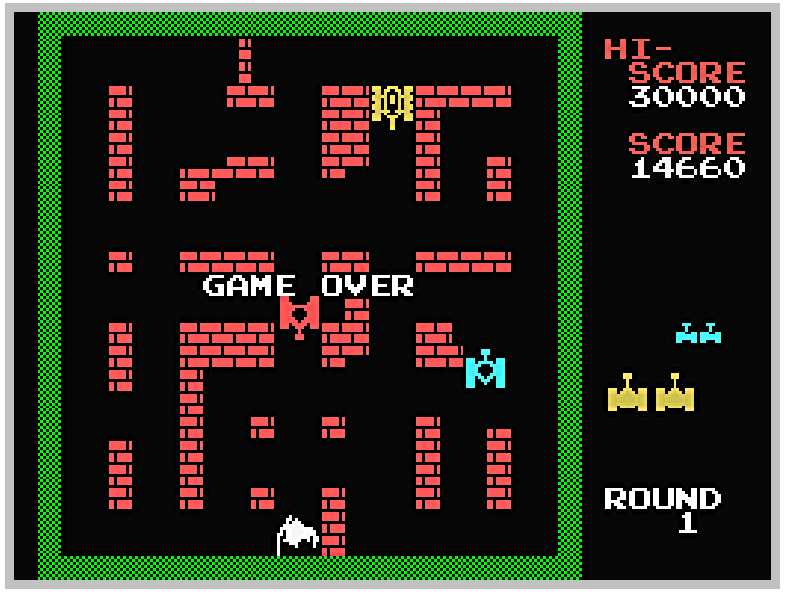


xgsun@fudan.edu.cn

孙晓光

2024-11-17

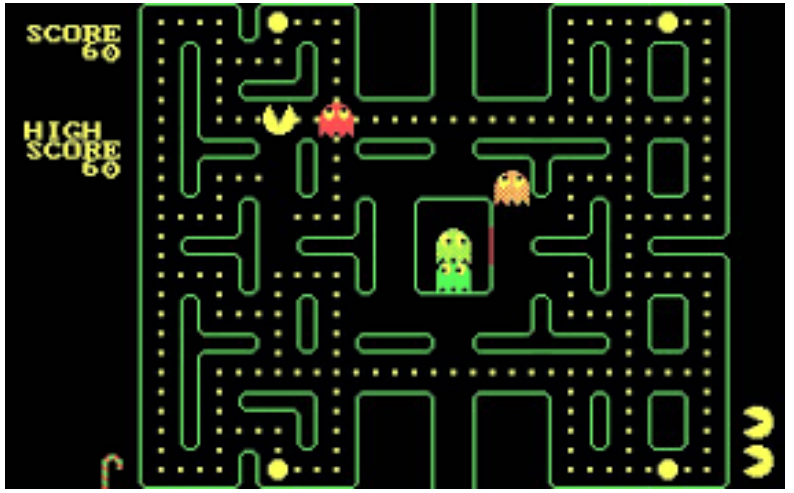
古老的街机游戏 arcade game



Tank Battalion

坦克大战

1980年风靡全球

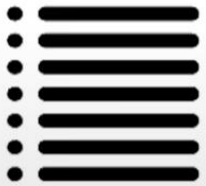


Pac-Man 吃豆人

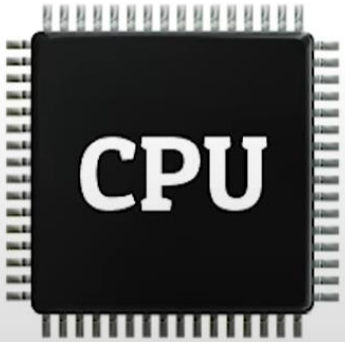
基于FPGA 的设计

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```

Compilation, Assembly, and Linking



Machine Code

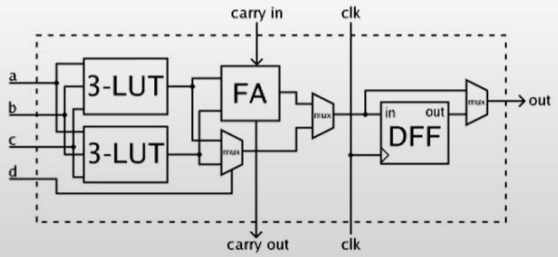



```
module Sound(
    input CLOCK_50,
    output reg SPEAKER
);

reg [25:0] count = 0;

always @(posedge CLOCK_50)
begin
    count <= count + 1'b1;

    if (count == 50000000 / 1000 / 2)
        begin
            SPEAKER <= ~SPEAKER;
            count <= 0;
        end
end

endmodule
```

a FPGA generally will run faster

Synthesis

没有代码！

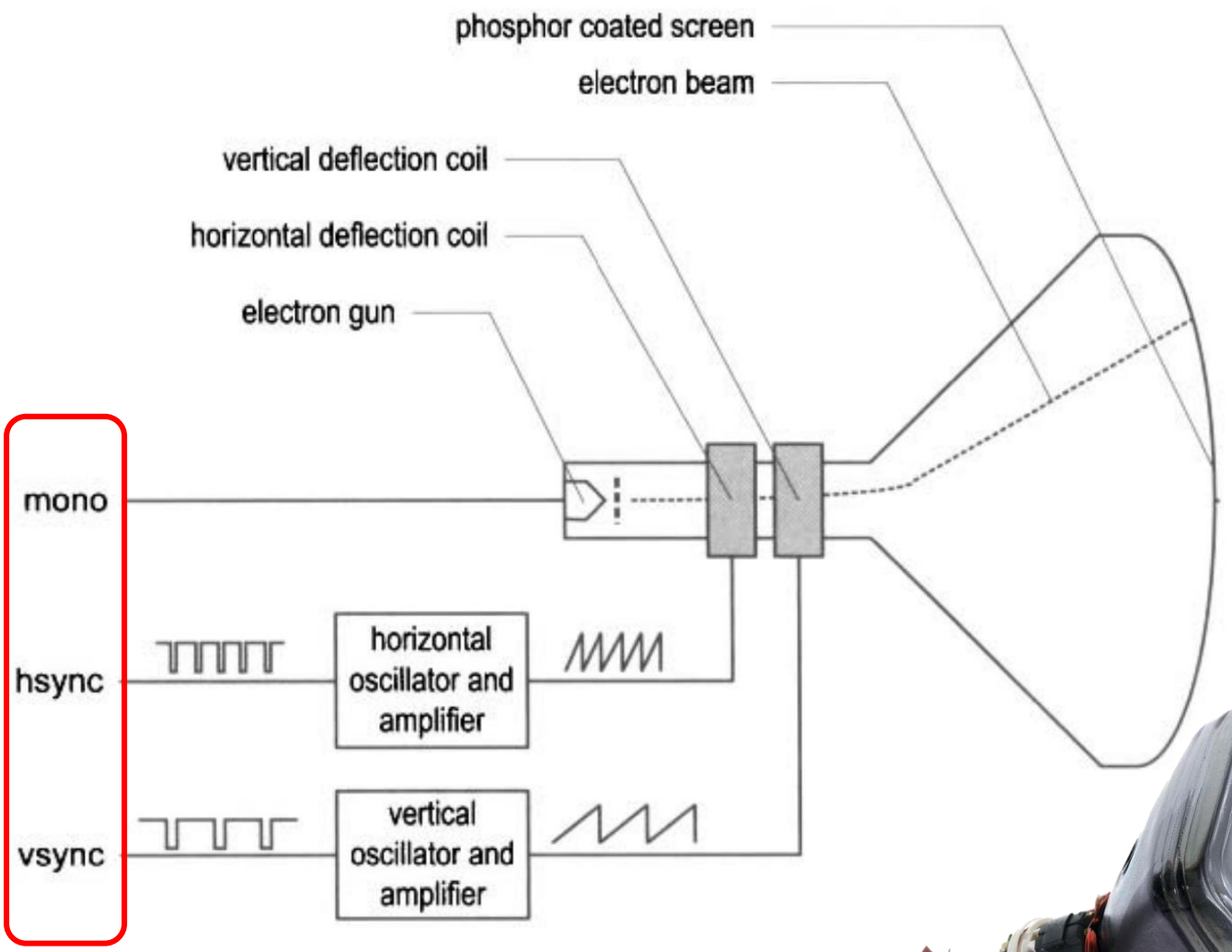


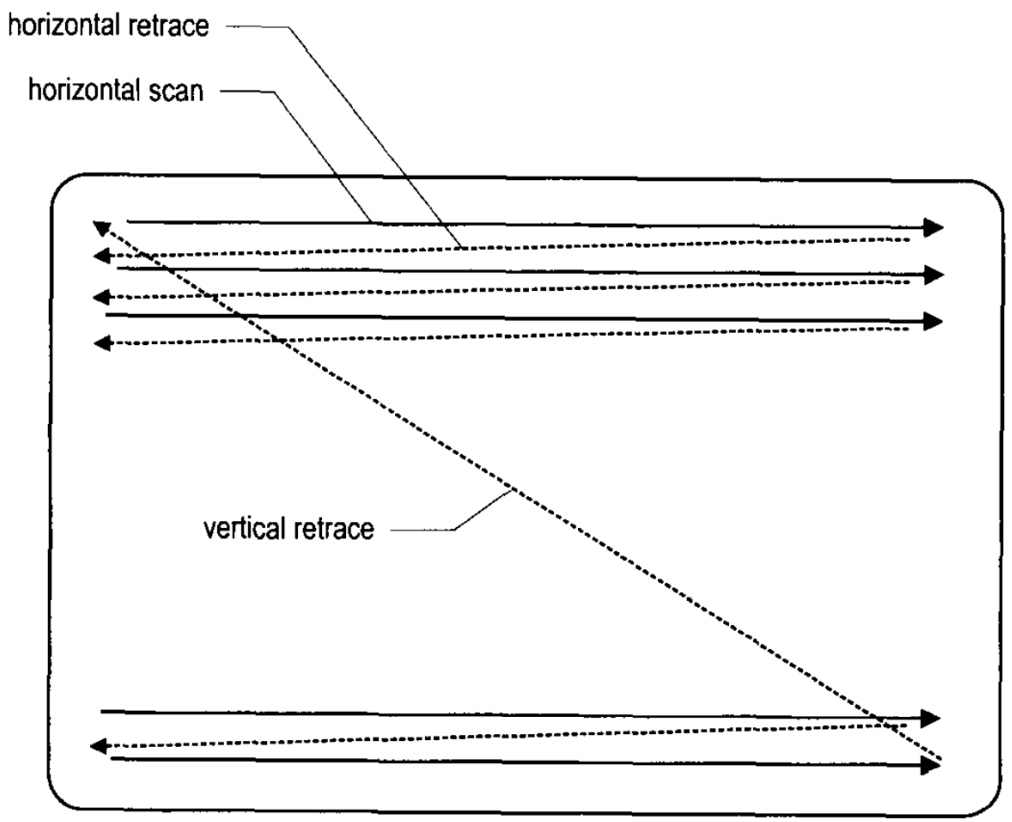
Logic Implementation

1

VGA

阴极射线管 CRT monitor





CRT scanning patter

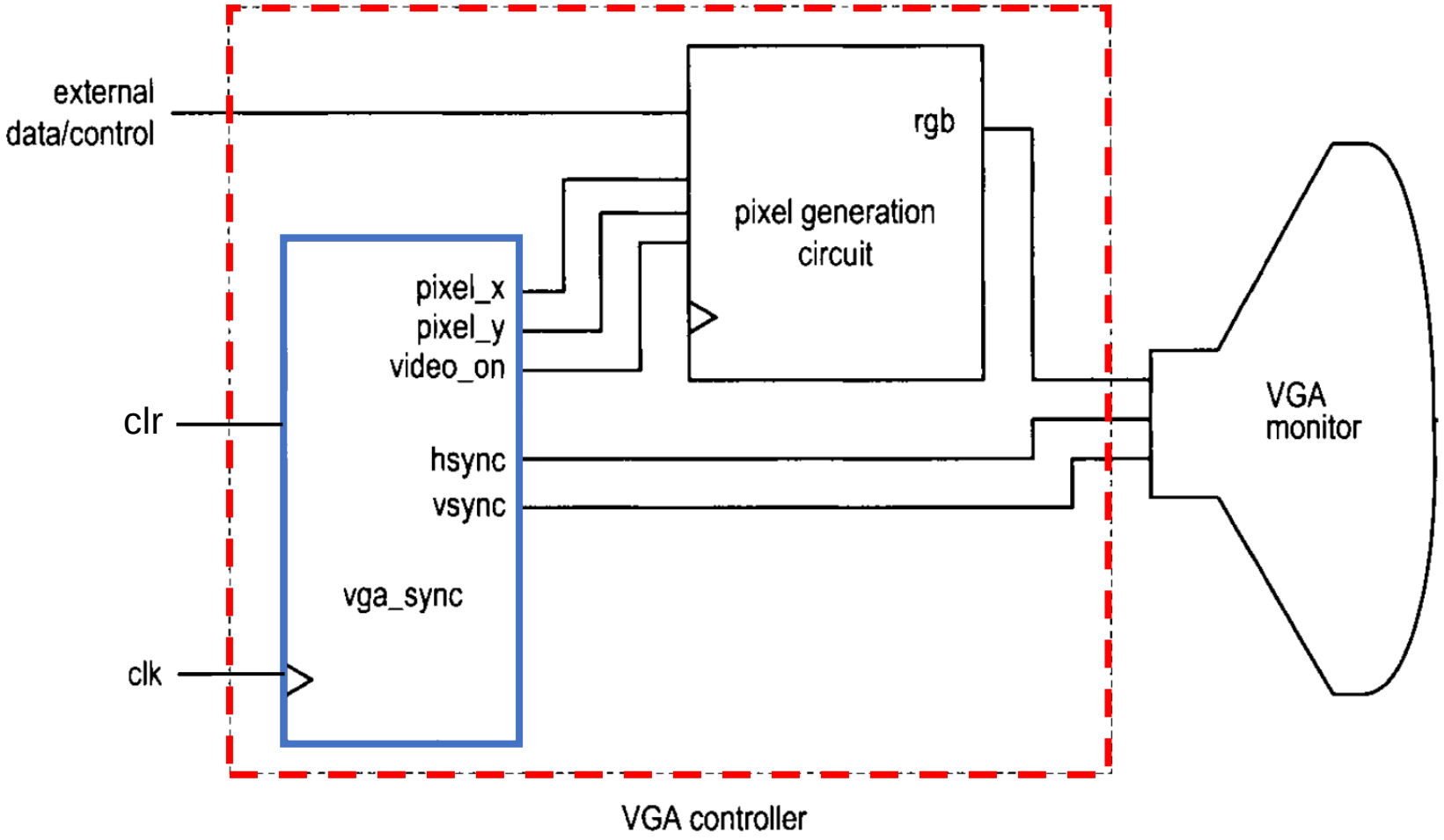
hsync信号：标志着**每一行的结束**，

vsync信号：标志着**整个帧的结束**。

【参考】教材P346，**VGA显示器**

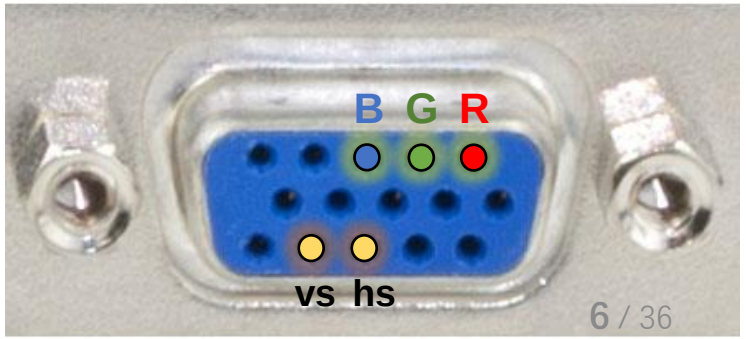
Video Graphics Array VGA 控制器

产生**同步信号** 和 串行输出**色彩数据**。



VGA cable

VGA connector



Nexys4 的 VGA 接口(约束文件)

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK100MHZ}];

##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { BTNC }]; #IO_L9P_T1_DQS_14 Sch=btnc

##VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[3] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]

set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[1] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[2] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[3] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]

set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }]; #IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[2] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[3] }]; #IO_L4P_T0_35 Sch=vga_b[3]

set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { VGA_HS }]; #IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { VGA_VS }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

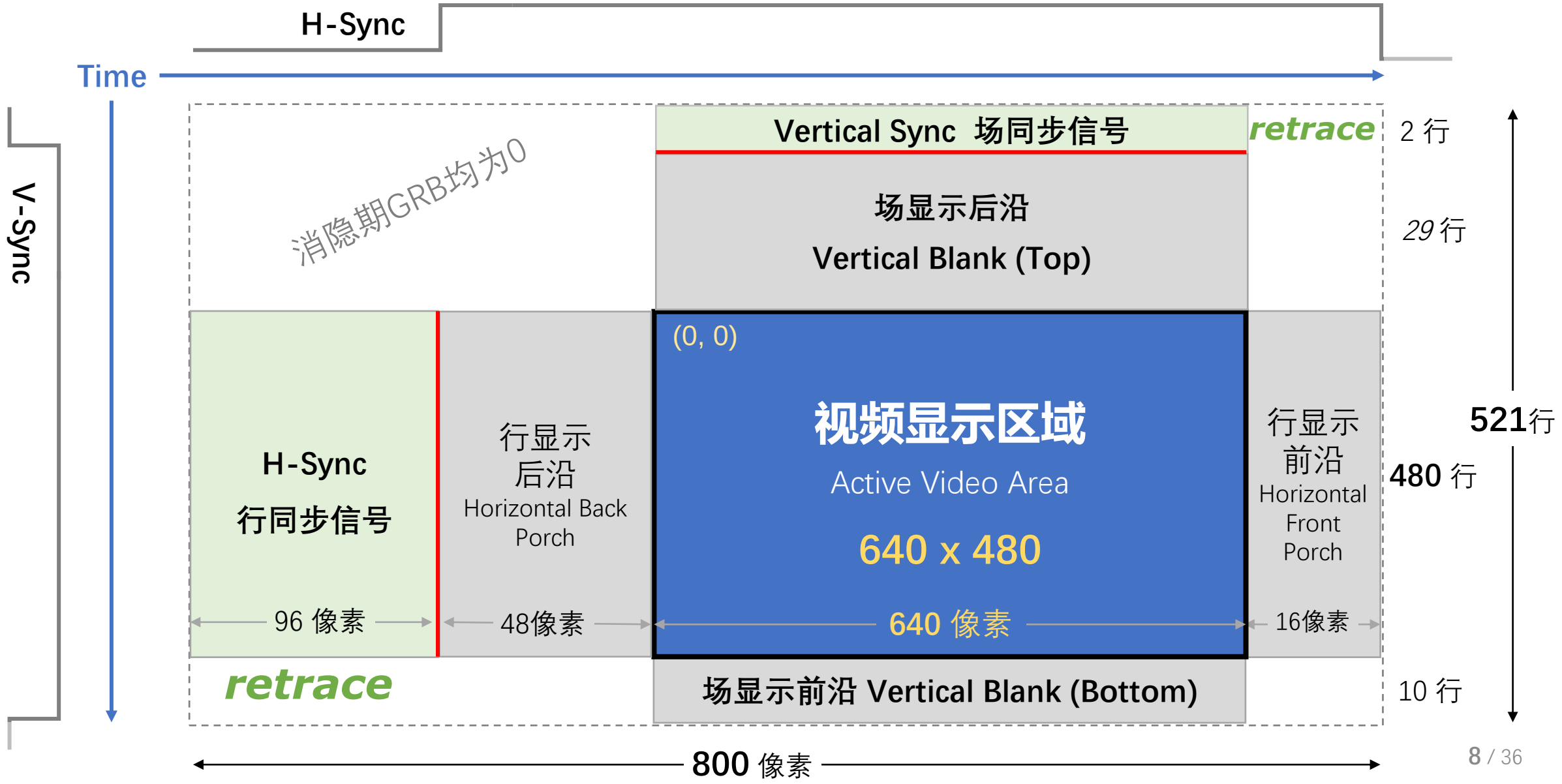
普通图像色彩：$2^8 * 2^8 * 2^8 = 2^{3*8} = 16{,}777{,}216$

色彩：$2^4 * 2^4 * 2^4 = 2^{3*4} = 4{,}096$

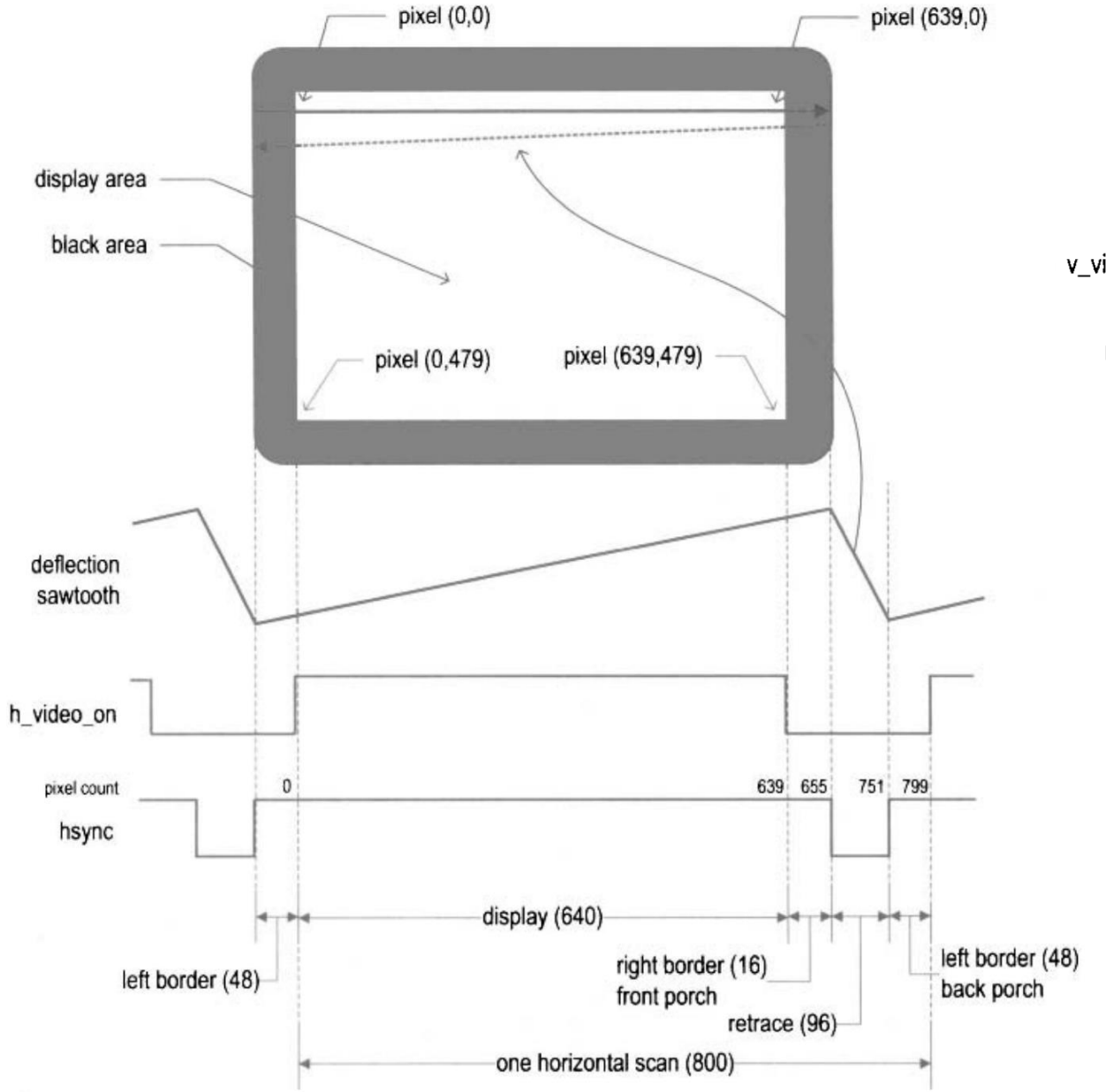
12 bits / pixel

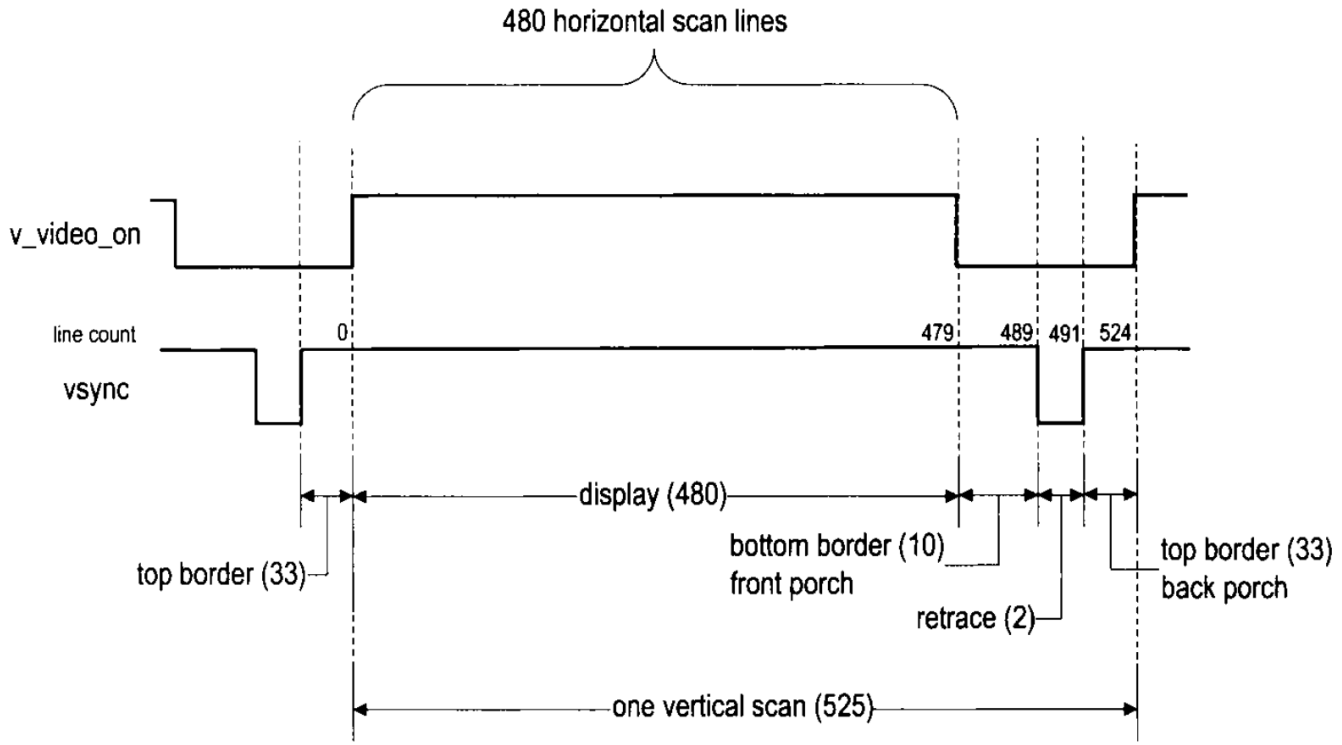
VGA原理 – 640x480@60Hz

$$60Hz * 521lines * 800pixels \approx 25MHz$$

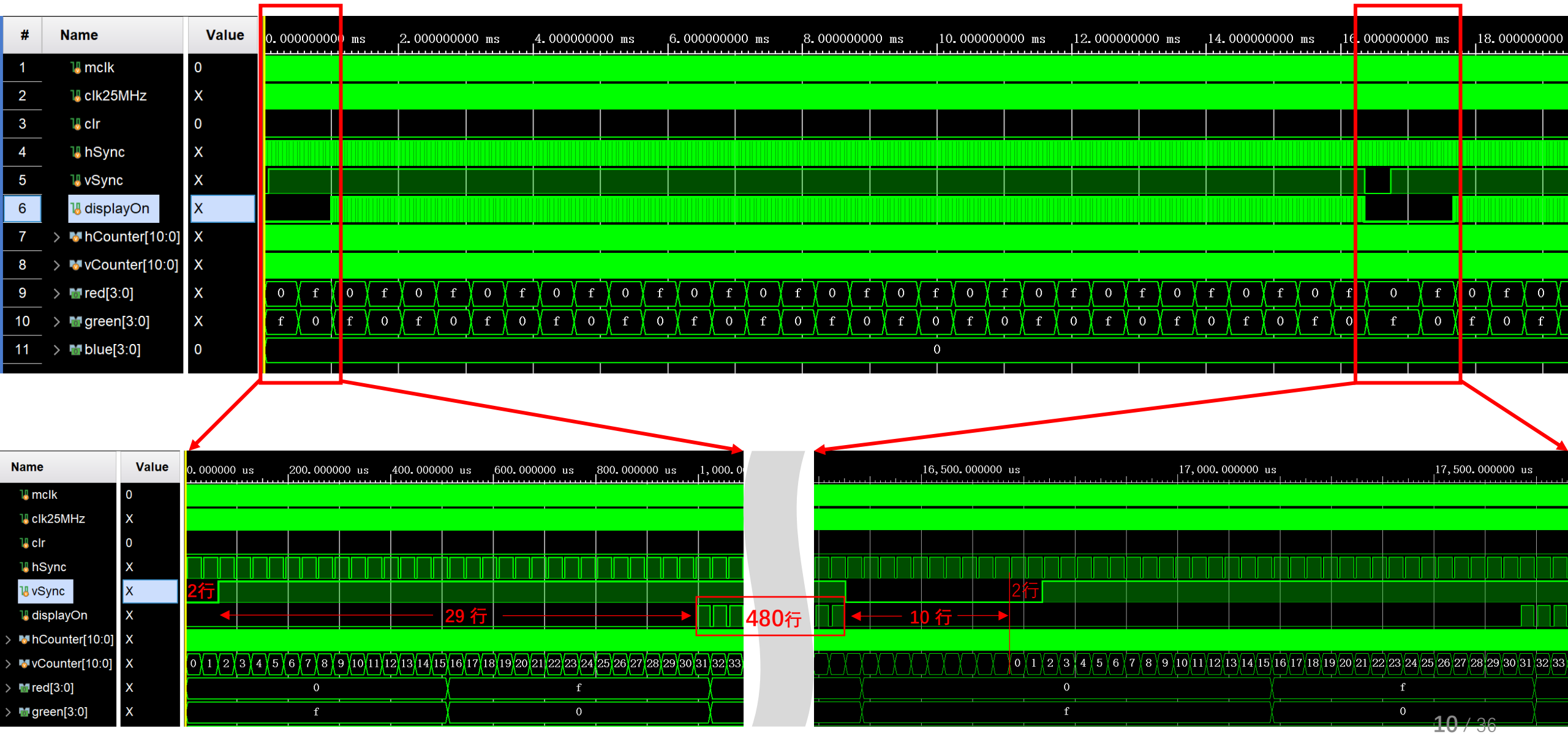


水平扫描、垂直扫描

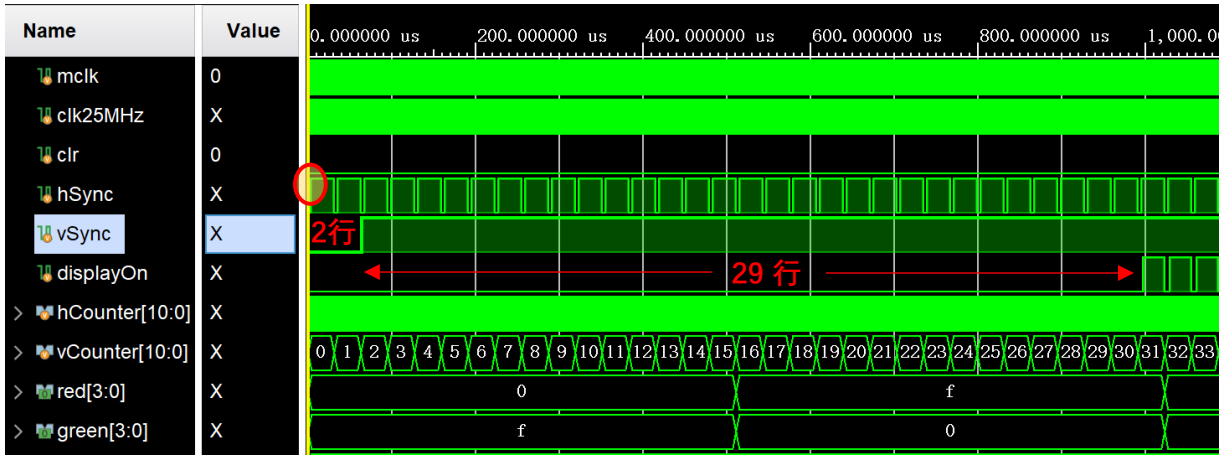

pixel (0,0)
pixel (639,0)
display area
black area
pixel (0,479)
pixel (639,479)
deflection
sawtooth
h_video_on
pixel count
0
639
655
751
799
hsync
display (640)
left border (48)
right border (16)
front porch
retrace (96)
left border (48)
back porch
one horizontal scan (800)


480 horizontal scan lines
v_video_on
line count
0
479
489
491
524
vsync
display (480)
top border (33)
bottom border (10)
front porch
retrace (2)
top border (33)
back porch
one vertical scan (525)

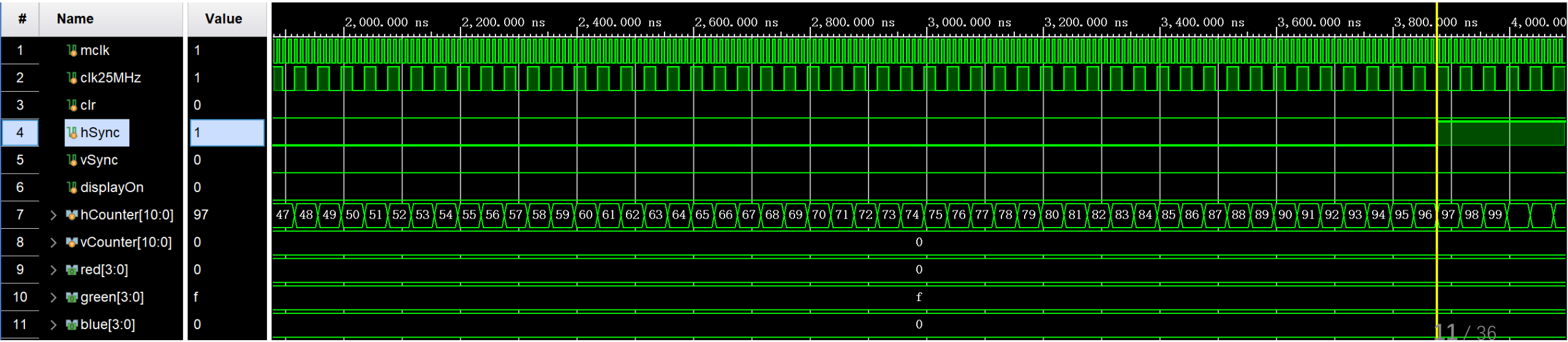
vSync 场同步（640x480）



hSync 行同步（640x480)

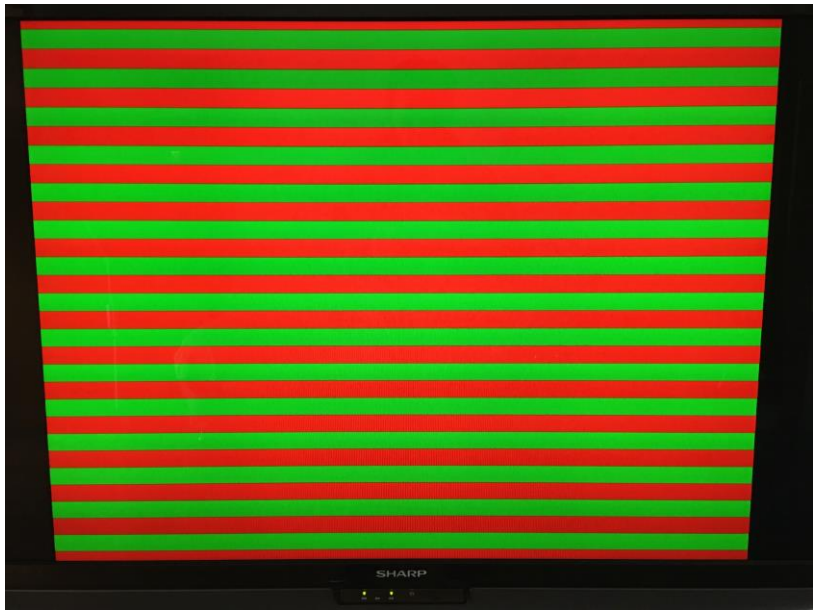


行同步信号需要：96个脉冲，96个像素

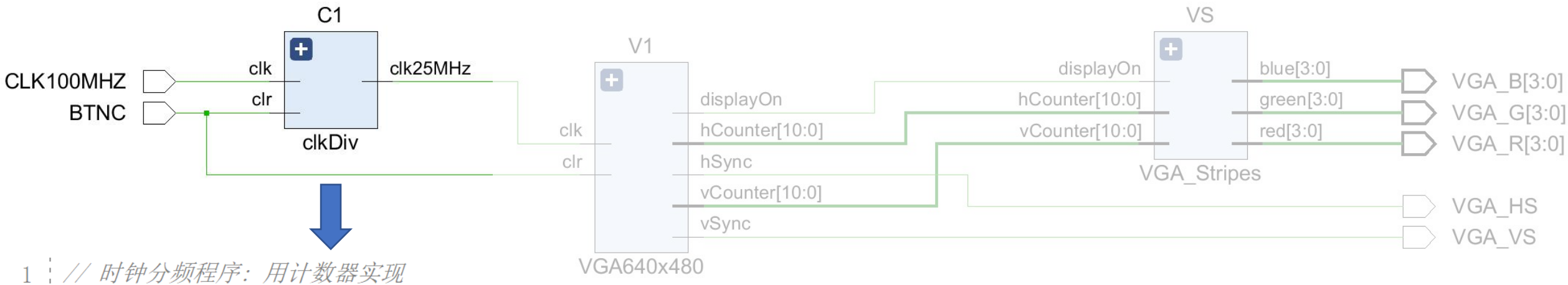


仿真代码

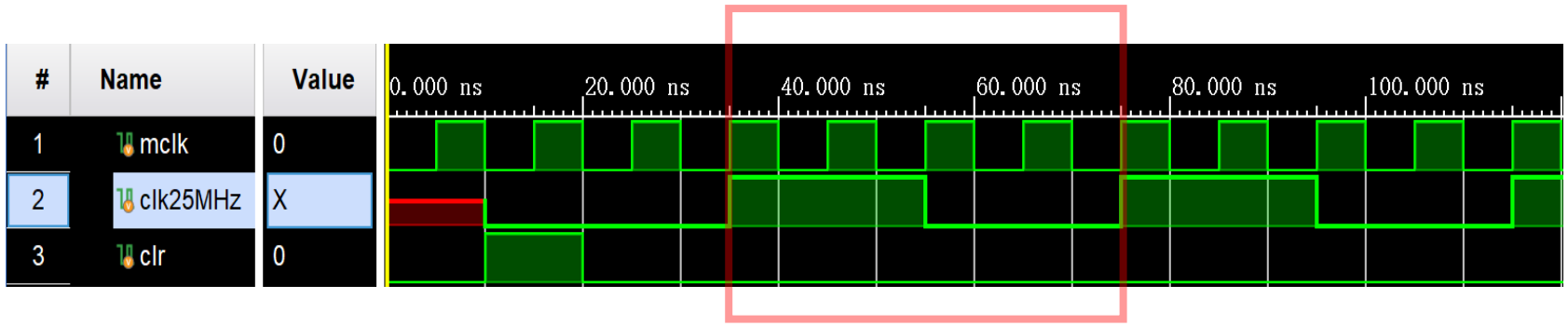
```
1  `timescale 1ns / 1ps
2  module VGA_Stripes_Sim( );
3      logic mclk, clr;    // 接口：激励信号
4      logic hSync, vSync; // 接口：输出信号
5      logic [3:0] Red, Green, Blue;
6
7      //局部变量
8      logic clk25MHz, displayOn;
9      logic [10:0] hCounter, vCounter;
10
11     always
12     begin   // 产生时钟信号
13         mclk = 0; #5; mclk = 1; #5;
14     end
15
16     initial
17     begin   // 提供初始激励信号
18         # 0; clr = 0;  #10; clr = 1;
19         #10; clr = 0;  hCounter = 0;  vCounter = 0;
20     end
22     // 实例化
23     clkDiv C1(.clk(mclk), .clr(clr), .clk25MHz(clk25MHz));
24
25     VGA640x480 V1(.clk(clk25MHz), .clr(clr),                  // Input
26                   .hSync(hSync), .vSync(vSync),               // Output ***
27                   .hCounter(hCounter), .vCounter(vCounter),// Output
28                   .displayOn(displayOn));                     // Output
29
30     VGA_Stripes VS(.displayOn(displayOn), .hCounter(hCounter), .vCounter(vCounter),
31                    .red(Red), .green(Green), .blue(Blue) ); // Output ***
32 endmodule
```



25MHz时钟分频模块

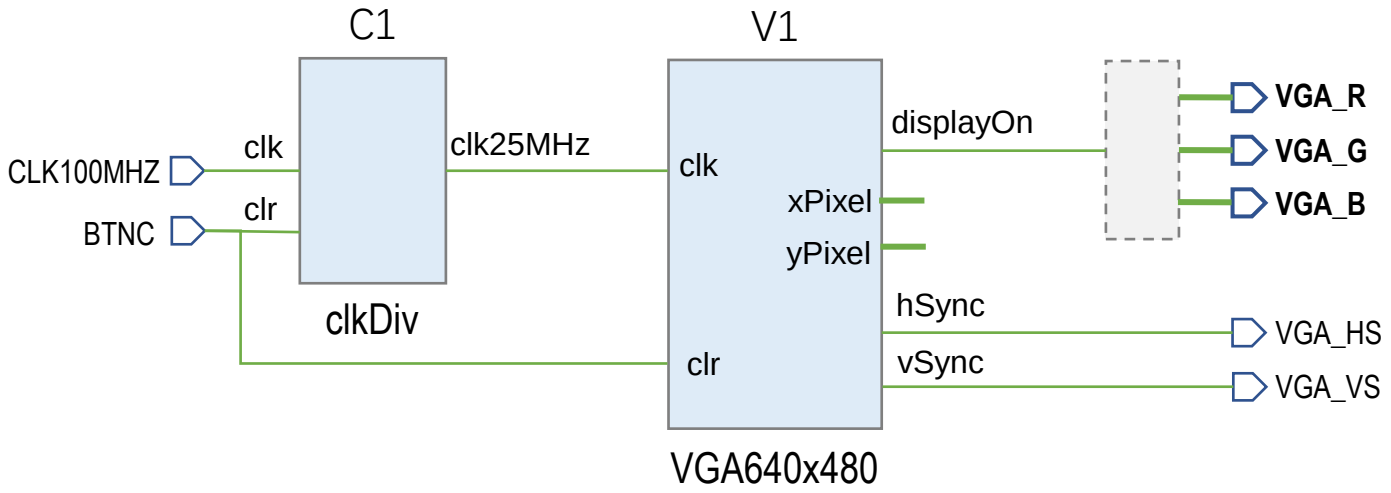



```
1  // 时钟分频程序：用计数器实现
2  module clkDiv(
3      input  logic clk,
4      input  logic clr,
5      output logic clk25MHz );
6
7      logic [1:0] q;  //reg
8      // 2-bit counter
9      always @(posedge clk, posedge clr)
10         if(clr==1) q <= 0;
11         else       q <= q + 1;
12
13     assign clk25MHz = q[1]; // 100/4=25MHz
14 endmodule
```



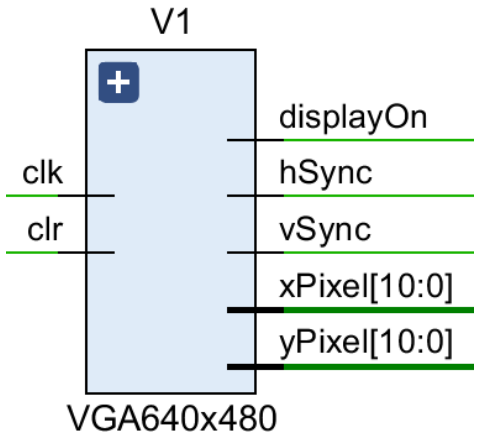
VGA 顶层文件

```
1  module VGA_Top(
2      input  logic CLK100MHZ, BTNC,
3      output logic VGA_HS, VGA_VS,
4      output logic [3:0] VGA_R, VGA_G, VGA_B );
5
6      logic clk25MHz, displayOn;
7      logic [10:0] xPixel, yPixel;
8
9      clkDiv C1(.clk(CLK100MHZ), .clr(BTNC), .clk25MHz(clk25MHz));
10
11     VGA640x480 V1(.clk(clk25MHz), .clr(BTNC),        // Input
12                   .hSync(VGA_HS), .vSync(VGA_VS),    // Output
13                   .xPixel(xPixel), .yPixel(yPixel),  // Output
14                   .displayOn(displayOn));            // Output
15
16     assign VGA_R = (displayOn) ? 4'b1111 : 4'b0000;
17     assign VGA_G = 4'b0;
18     assign VGA_B = 4'b0;
19 endmodule
```



VGA 模块（640x480）

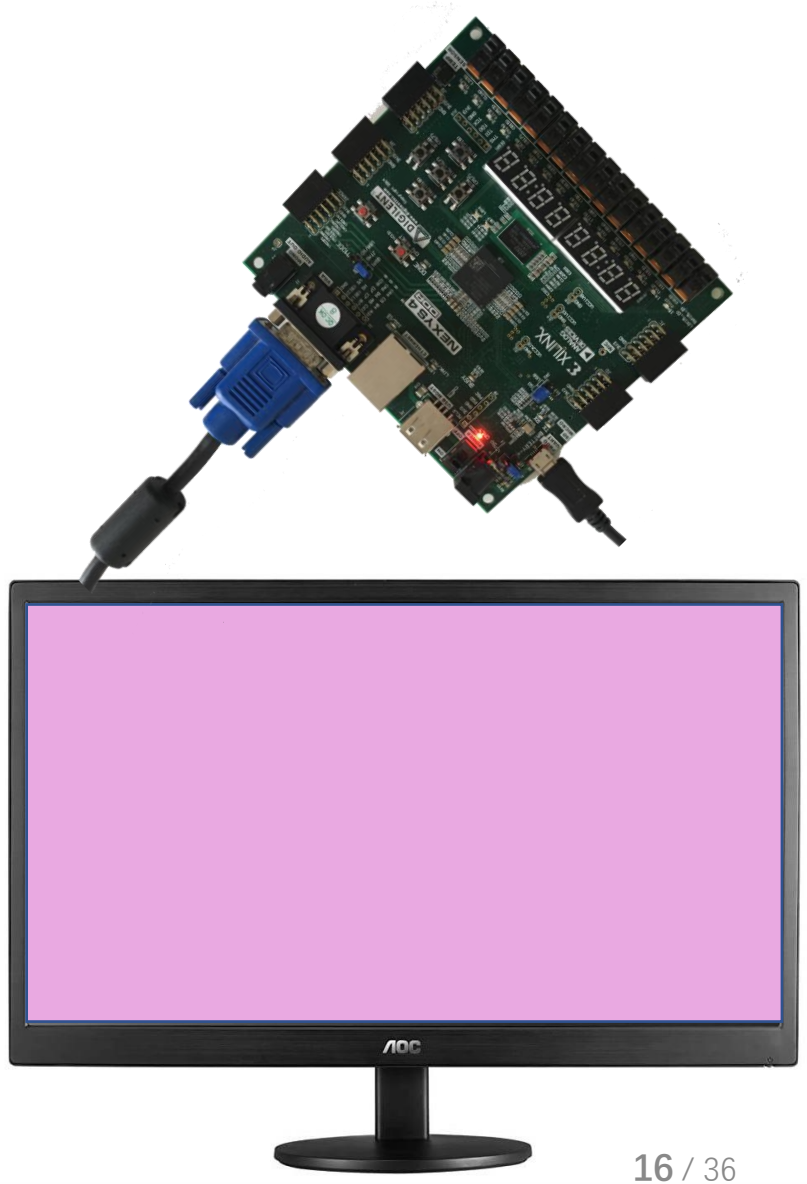
```
module VGA640x480(  input  logic clk, clr,
    output logic hSync, vSync,     //行同步信号, 场同步信号
    output logic [10:0] xPixel, yPixel, //行(800)、列(521)计数器
    output logic displayOn );      //是否处于可显示的范围?

    // horizontal constants
    localparam H_DISPLAY = 640; // horizontal display width
    localparam H_SYNC    =  96; // horizontal sync width
    localparam H_BACK    =  48; // left border (back porch)
    localparam H_FRONT   =  16; // right border (front porch)
    // vertical constants
    localparam V_DISPLAY = 480; // vertical display height
    localparam V_SYNC    =   2; // vertical sync lines
    localparam V_TOP     =  29; // vertical top border
    localparam V_BOTTOM  =  10; // vertical bottom border
    // derived constants
    localparam
      H_SYNC_START = H_SYNC + H_BACK,                    //行显示后沿 = 144(96+48)
      H_SYNC_END   = H_SYNC + H_BACK + H_DISPLAY,        //行显示前沿 = 784(96+48+640)
      H_PIXELS     = H_SYNC + H_BACK + H_DISPLAY + H_FRONT, //行像素点数 = 800(96+48+640+16)
      V_SYNC_START = V_SYNC + V_TOP,                     //场显示后沿 =  31(2+29)
      V_SYNC_END   = V_SYNC + V_TOP + V_DISPLAY,         //场显示前沿 = 511(2+29+480)
      V_LINES      = V_SYNC + V_TOP + V_BOTTOM + V_DISPLAY; //    总行数 = 521(2+29+480+10)

    logic hMaxed, vMaxed;
    assign hMaxed = (xPixel == H_PIXELS-1) || clr;  // set when xPixel is maximum
    assign vMaxed = (yPixel == V_LINES -1) || clr;  // set when yPixel is maximum
```




```
    // horizontal position counter
    always @(posedge clk)
    begin
        //行同步信号  [96~784)
        hSync <= (xPixel>=H_SYNC && xPixel<H_SYNC_END);
        if(hMaxed)  xPixel <= 0;
        else        xPixel <= xPixel + 1;
    end

    // vertical position counter
    always @(posedge clk)
    begin
        //场同步信号 [2~511)
        vSync <= (yPixel>=V_SYNC && yPixel<V_SYNC_END);
        if(hMaxed)
            if (vMaxed)  yPixel <= 0;
            else         yPixel <= yPixel + 1;
    end

    //Enable displayOn when beam is in "safe" visible frame
    assign displayOn = ((xPixel>=H_SYNC_START) && // [144~
                        (xPixel< H_SYNC_END) &&   //  784)
                        (yPixel>=V_SYNC_START) && // [31~
                        (yPixel< V_SYNC_END));    //  511)
endmodule
```

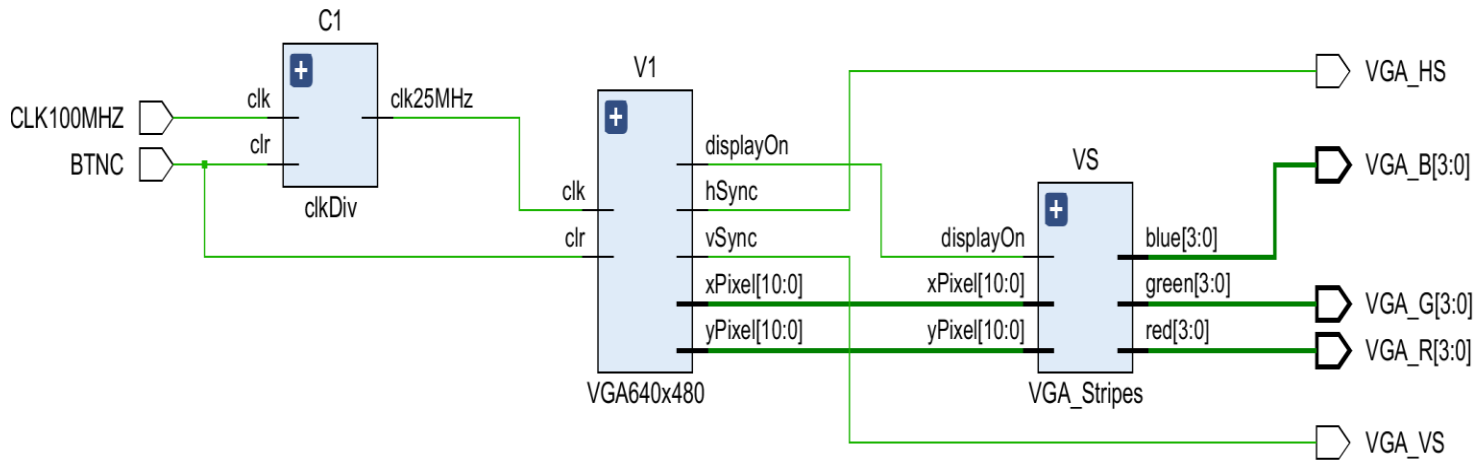
VGA -colors 顶层文件

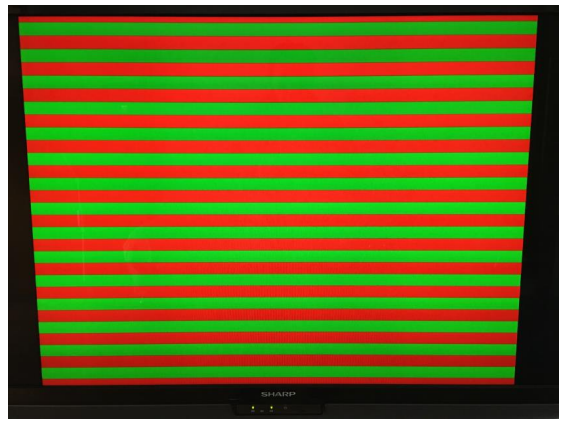
```
1  module VGA_colors_Top(
2      input  logic CLK100MHZ, BTNC,
3      input  logic [11:0] SW,
4      output logic VGA_HS, VGA_VS,
5      output logic [3:0] VGA_R, VGA_G, VGA_B );
6
7      logic clk25MHz, displayOn;
8      logic [10:0] xPixel, yPixel;
9
10     clkDiv C1(.clk(CLK100MHZ), .clr(BTNC), .clk25MHz(clk25MHz));
11
12     VGA640x480 V1(.clk(clk25MHz), .clr(BTNC),       // Input
13                   .hSync(VGA_HS), .vSync(VGA_VS),   // Output
14                   .xPixel(xPixel), .yPixel(yPixel), // Output
15                   .displayOn(displayOn));           // Output
16
17     assign VGA_R = (displayOn) ? SW[11:8] : 4'b0000;
18     assign VGA_G = (displayOn) ? SW[ 7:4] : 4'b0000;
19     assign VGA_B = (displayOn) ? SW[ 3:0] : 4'b0000;
20 endmodule
```



VGA -stripes顶层文件

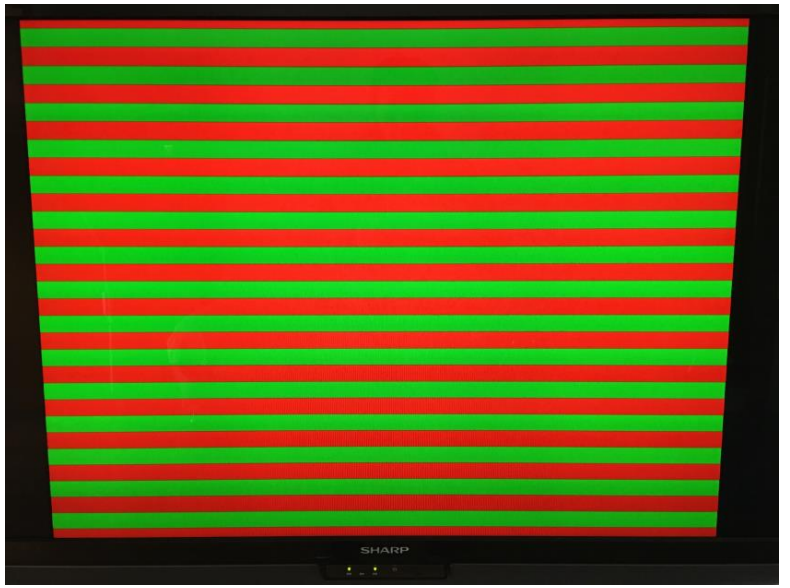
```
1  module VGA_Stripes_Top(
2      input  logic CLK100MHZ, BTNC,
3      output logic VGA_HS, VGA_VS,
4      output logic [3:0] VGA_R, VGA_G, VGA_B );
5
6      logic clk25MHz, displayOn;
7      logic [10:0] xPixel, yPixel;
8
9      clkDiv C1(.clk(CLK100MHZ), .clr(BTNC), .clk25MHz(clk25MHz));
10
11     VGA640x480 V1(.clk(clk25MHz), .clr(BTNC),          // Input
12                   .hSync(VGA_HS), .vSync(VGA_VS),     // Output ***
13                   .xPixel(xPixel), .yPixel(yPixel),   // Output
14                   .displayOn(displayOn));             // Output
15
16     VGA_Stripes VS(.displayOn(displayOn),
17                    .xPixel(xPixel), .yPixel(yPixel),          // Input
18                    .red(VGA_R), .green(VGA_G), .blue(VGA_B)); // Output ***
19 endmodule
```





VGA – stripes模块

```
1  module VGA_Stripes(
2      input  logic displayOn,
3      input  logic [10:0] xPixel, yPixel,
4      output logic [3:0]  red, green, blue );
5
6      //  =====  横彩条  ======
7      assign red   =  {4{yPixel[4]}}; 32像素
8      assign green = ~{4{yPixel[4]}}; 15条绿色
9      assign blue  = 0;
10
11 //   //  =====  竖彩条  ======
12 //   assign red   =  {4{xPixel[4]}};
13 //   assign green = ~{4{xPixel[4]}};
14 //   assign blue  = 0;
15 endmodule
```



不同频率的时钟？

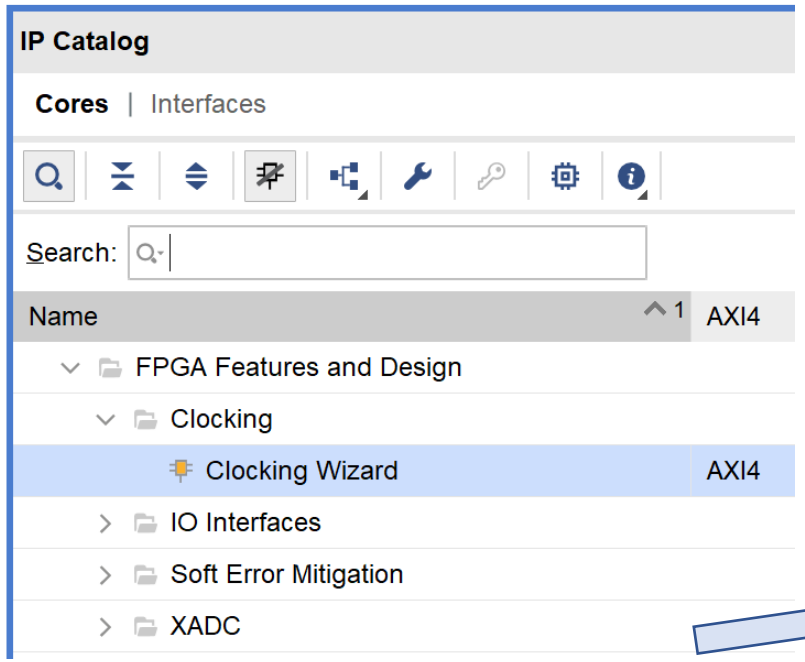
时钟管理块 CMT
(**C**lock **M**anagement **T**iles)

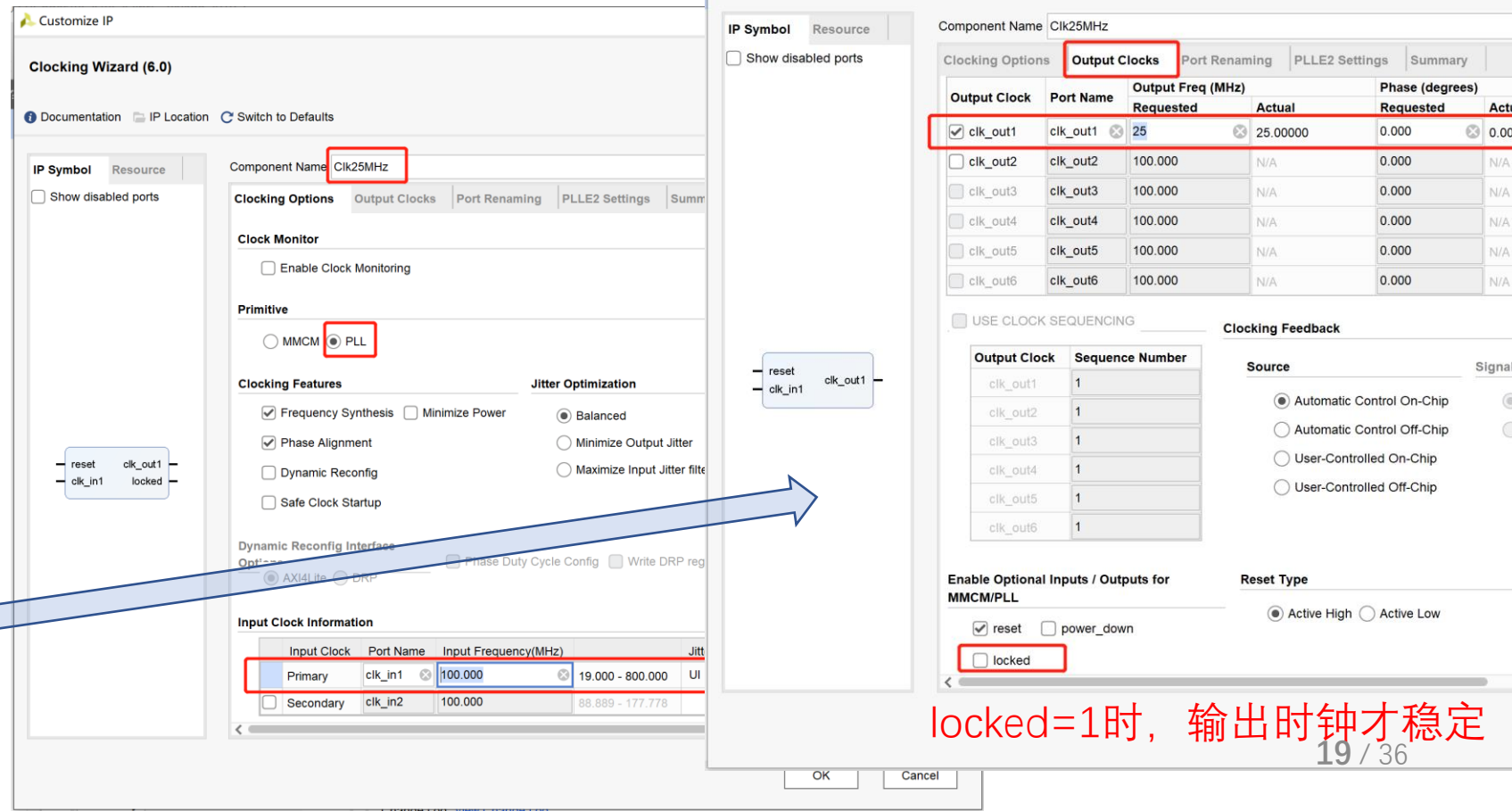
提供时钟频率合成、倾斜矫正和抖动滤波功能

混合时钟管理器 MMCM
(**M**ixed-**M**ode **C**lock **M**anager)

锁相环 PLL
(**P**hase-**L**ocked **L**oop)

- 可生成多个时钟
- 任意分频、倍频， 如65MHz, 200MHz…





locked=1时，输出时钟才稳定

用PLL时钟

```
1  module VGA_clkIP_Top(
2      input  logic CLK100MHZ, BTNC,
3      output logic VGA_HS, VGA_VS,
4      output logic [3:0] VGA_R, VGA_G, VGA_B   );
5
6      logic clk25MHz, displayOn;
7      logic [10:0] xPixel, yPixel;
8
9      Clk25MHz C1(.clk_out1(clk25MHz),                    // Output
10                 .reset(BTNC), .clk_in1(CLK100MHZ));
11
12     VGA640x480 V1(.clk(clk25MHz), .clr(BTNC),           // Input
13                   .hSync(VGA_HS), .vSync(VGA_VS),       // Output
14                   .xPixel(xPixel), .yPixel(yPixel),     // Output
15                   .displayOn(displayOn));               // Output
16
17     assign VGA_R = (displayOn) ? 4'b1111 : 4'b0000;
18     assign VGA_G = 4'b0;
19     assign VGA_B = 4'b0;
20 endmodule
```

【作业1】SVGA 显示控制器设计
(Super VGA)

SVGA 800x600 pixels @72Hz

- pixel rate: **50** MHz
- horizontal display region: **800** pixels
- horizontal retrace: 120 pixels
- horizontal back porch: 56 pixels
- horizontal front porch: 64 pixels
- vertical display region: **600** lines
- vertical retrace: 6 lines
- vertical top: 37 lines
- vertical bottom: 23 lines

SVGA 800x600 pixels @60Hz

- pixel rate: **40** MHz
- horizontal display region: **800** pixels
- horizontal retrace: 128 pixels
- horizontal back porch: 88 pixels
- horizontal front porch: 40 pixels
- vertical display region: **600** lines
- vertical retrace: 4 lines
- vertical top: 23 lines
- vertical bottom: 1 lines

【作业1】XGA 显示控制器设计

(Extended Graphics Array)

XGA 1024x768 pixels @60Hz

- pixel rate: **65** MHz (也可用**50**MHz代替)
- horizontal display region: **1024** pixels
- horizontal retrace: 136 pixels
- horizontal back porch: 160 pixels
- horizontal front porch: 24 pixels
- vertical display region: **768** lines
- vertical retrace: 6 lines
- vertical top: 29 lines
- vertical bottom: 3 lines

XGA 1280x1024 pixels @60Hz

- pixel rate: **108** MHz
- horizontal display region: **1280** pixels
- horizontal retrace: 112 pixels
- horizontal back porch: 248 pixels
- horizontal front porch: 48 pixels
- vertical display region: **1024** lines
- vertical retrace: 3 lines
- vertical top: 38 lines
- vertical bottom: 1 lines

2

Picture

如何显示图像？

①Object-mapped：对于简单图形，直接在指定区域显示色彩。

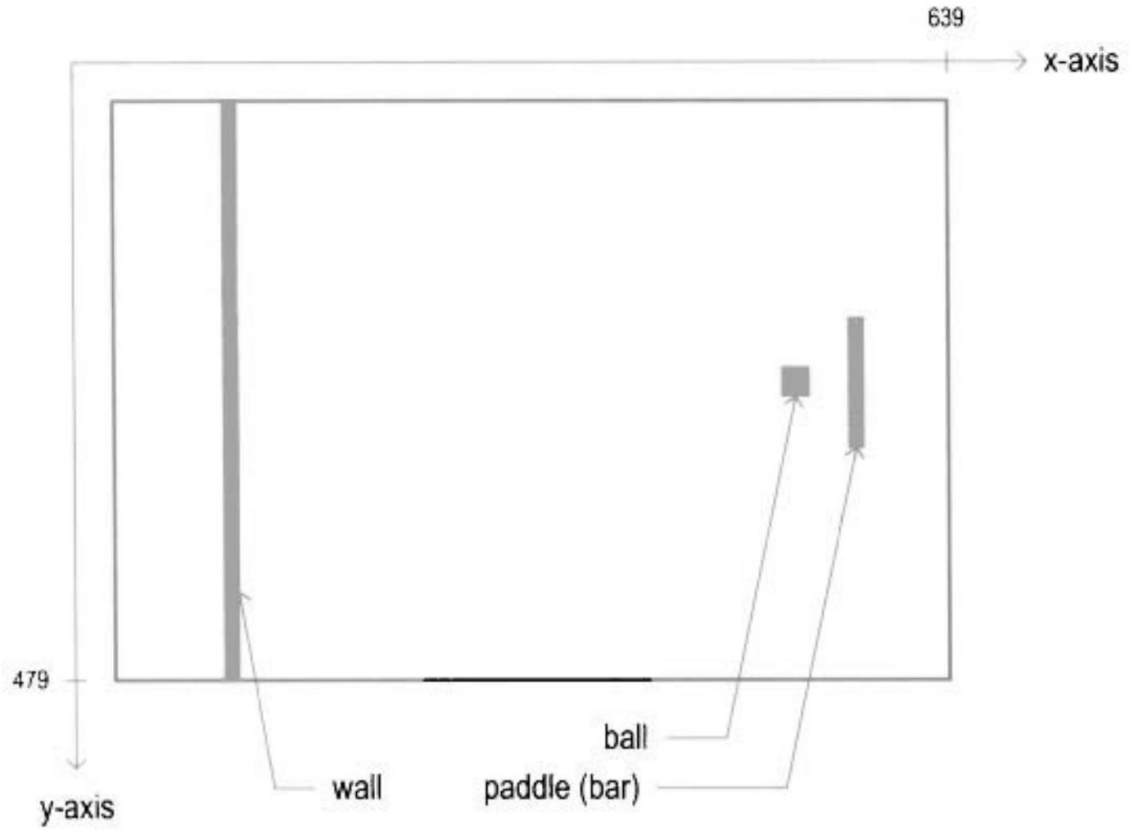
② Bit-mapped：对于图片，将图片每个像素映射到屏幕上，
如果图片较大，则需要大量的内存资源。

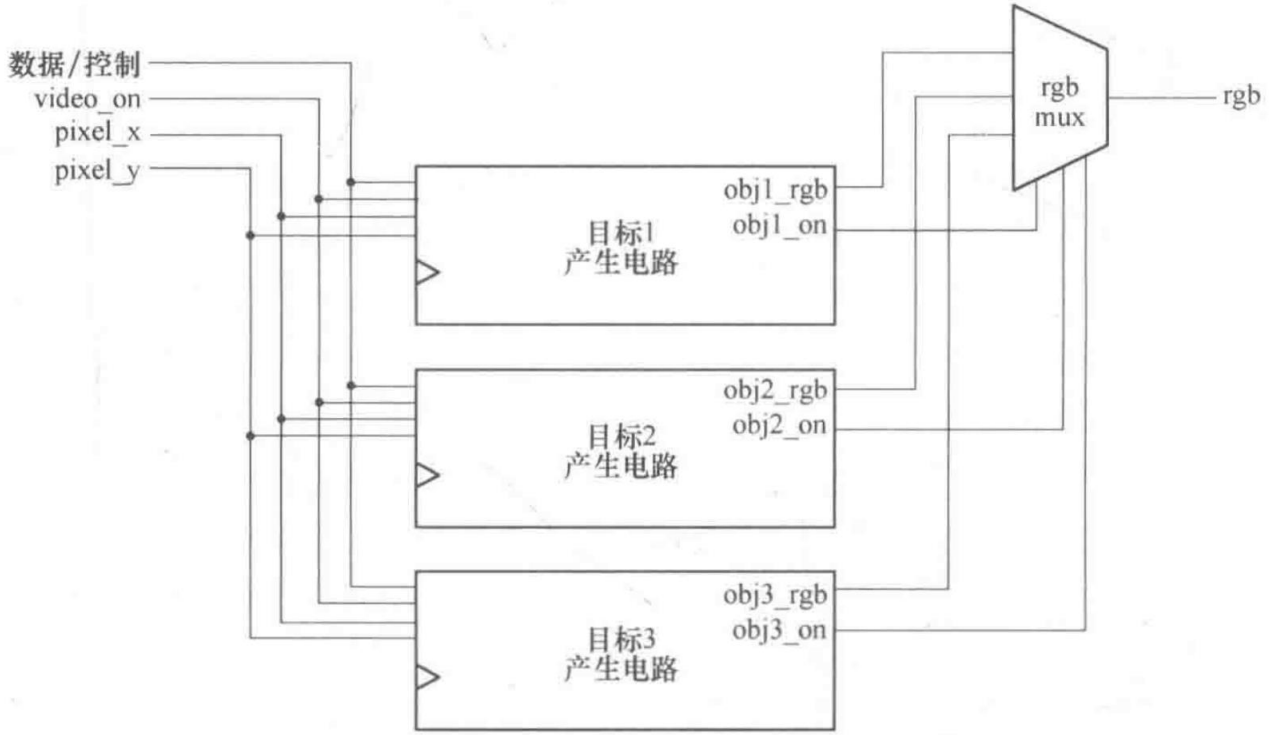
③ Tile-mapped：为减少内存需求，将重复使用的内容做成**小块**，
如，ASCII字符，分别做成16x8点阵字库。

① Object-mapped

静止

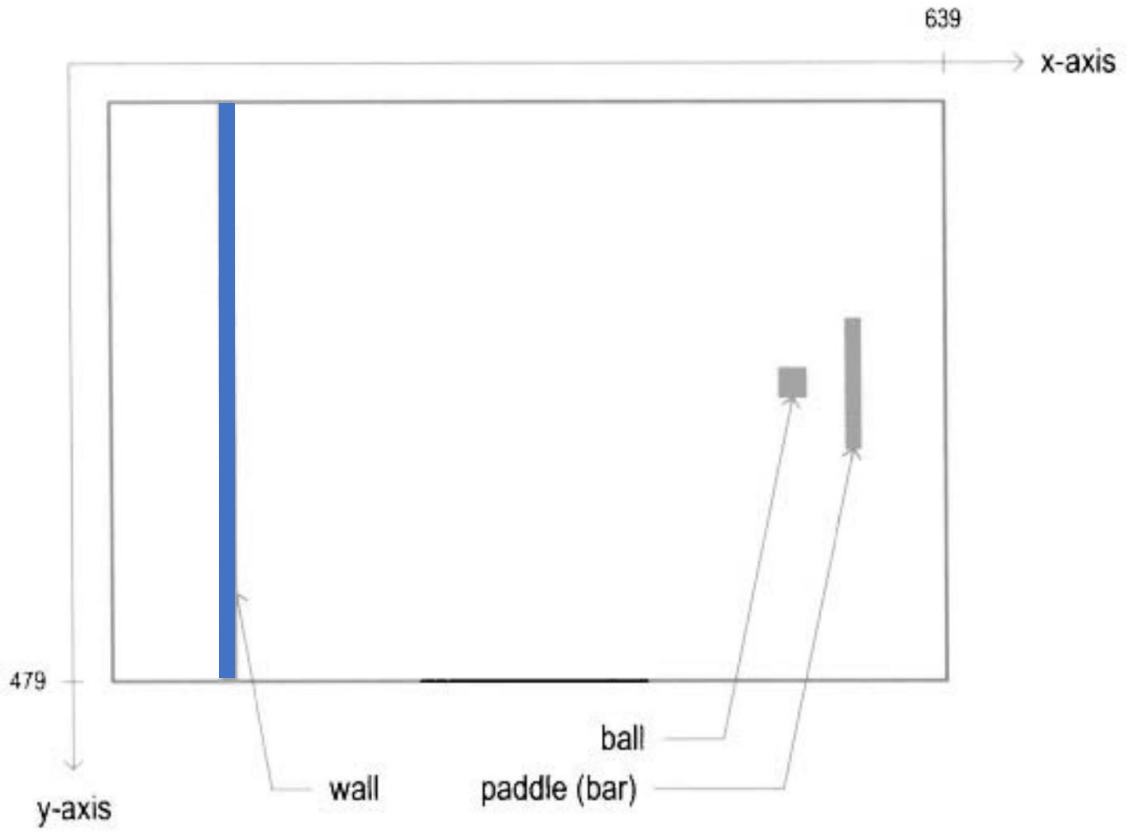
对于简单几何图形，直接在指定区域显示颜色。





乒乓球：wall

静止

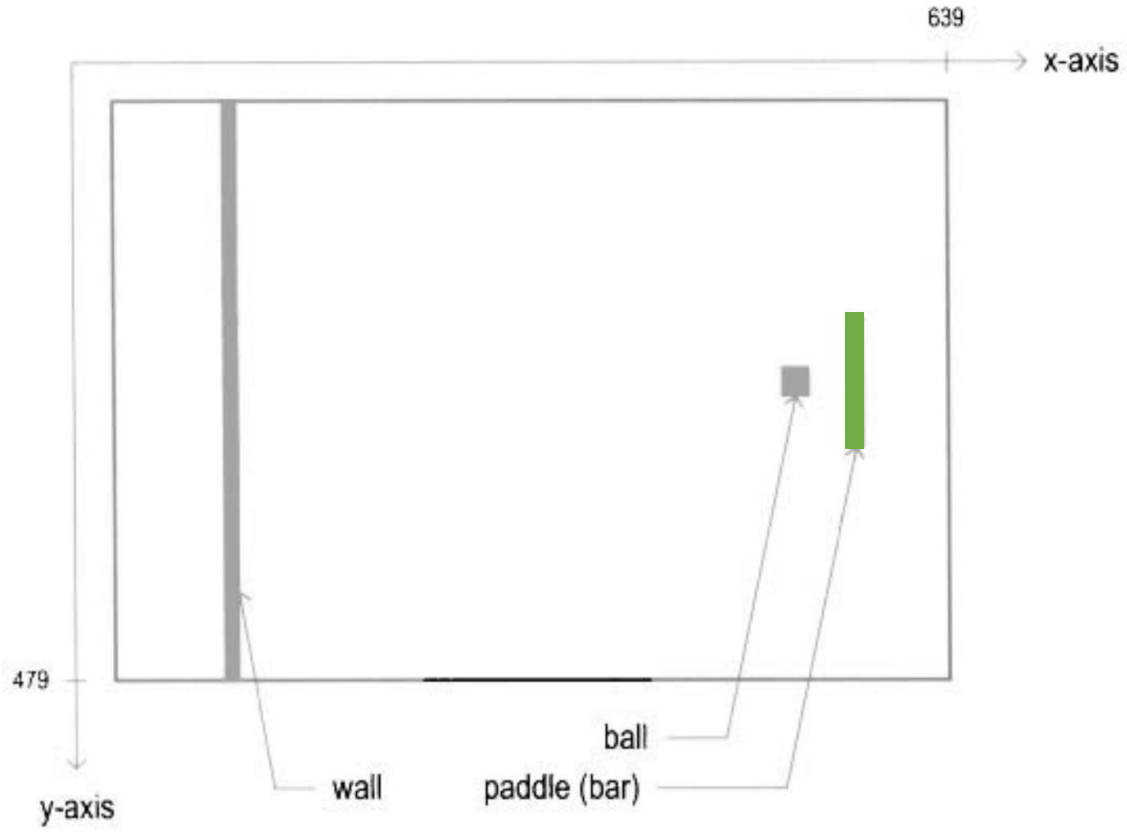



```
localparam WALL_X_L = H_SYNC_START + 30; // left boundary
localparam WALL_X_R = H_SYNC_START + 40; // right boundary

// (wall) left vertical strip: pixel within wall
assign wallOn = (WALL_X_L <= pix_x) && (pix_x <= WALL_X_R);
// wall rgb output
assign wall_r = 4'b0000;
assign wall_g = 4'b0000;
assign wall_b = 4'b1111; // blue
```

```
else if (wallOn)
begin
    red   = wall_r;
    green = wall_g;
    blue  = wall_b;
end
```

乒乓球：bar（球拍）

静止

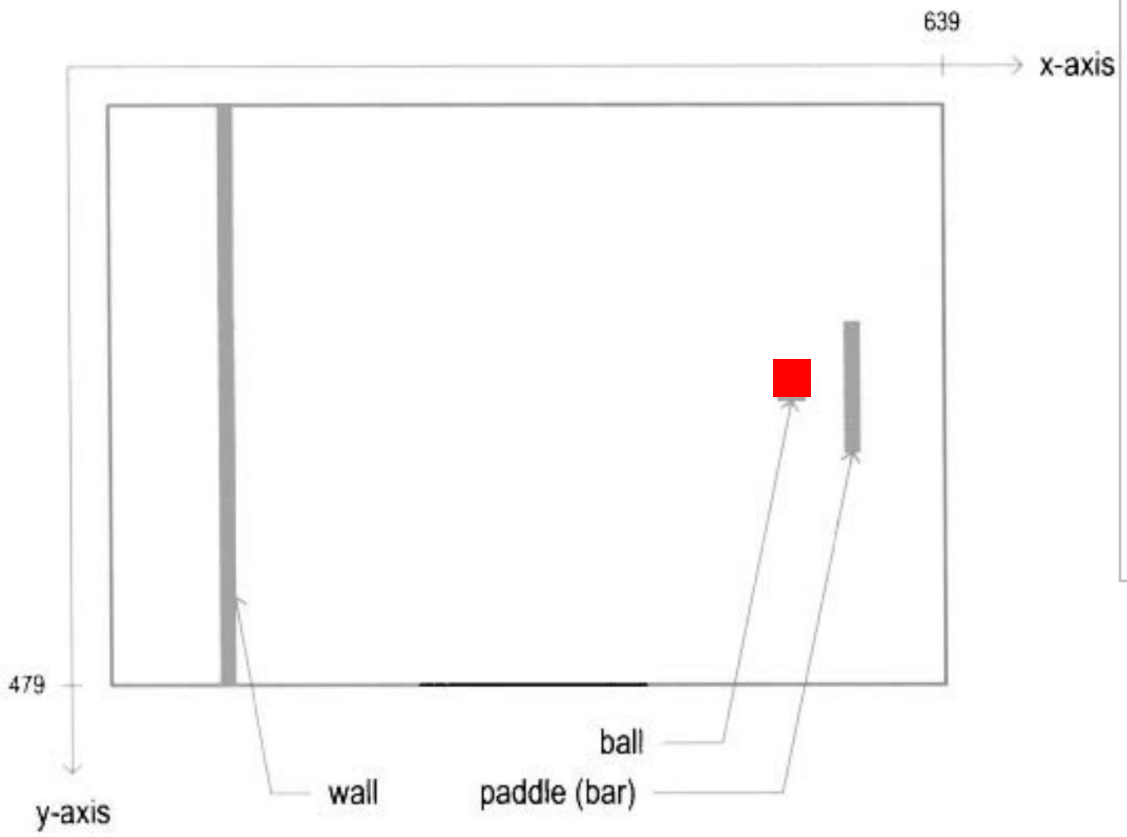



```
// right vertical === bar ===
localparam BAR_X_L = H_SYNC_START + 600; // left boundary
localparam BAR_X_R = H_SYNC_START + 605; // right boundary
localparam BAR_Y_SIZE = 70;              // bar 的高度
localparam BAR_Y_T = V_SYNC_START + MAX_Y/2 - BAR_Y_SIZE/2; //Top=204
localparam BAR_Y_B = BAR_Y_T + BAR_Y_SIZE -1; // bottom boundary

// (bar) :  pixel within bar
assign barOn = ((BAR_X_L<=pix_x) && (pix_x<=BAR_X_R) &&
                (BAR_Y_T<=pix_y) && (pix_y<=BAR_Y_B));
assign bar_r = 4'b0000;
assign bar_g = 4'b1111; // green
assign bar_b = 4'b0000;
```

```
else if (barOn)
begin
    red   = bar_r;
    green = bar_g;
    blue  = bar_b;
end
```

乒乓球：ball（方球）

静止




```
// === ball ===
BALL_SIZE = 8,
BALL_X_L = H_SYNC_START + 580,        // left boundary
BALL_X_R = BALL_X_L + BALL_SIZE - 1, // right boundary
BALL_Y_T = V_SYNC_START + MAX_Y/2 -BALL_SIZE/2,//top boundary
BALL_Y_B = BALL_Y_T + BALL_SIZE - 1;        // bottom boundary

// (ball): pixel within squared ball
assign ballOn = ((BALL_X_L<=pix_x) && (pix_x<=BALL_X_R) &&
                 (BALL_Y_T<=pix_y) && (pix_y<=BALL_Y_B));
assign ball_r = 4'b1111; // red
assign ball_g = 4'b0000;
assign ball_b = 4'b0000;
```

```
else if (ballOn)
begin
    red   = ball_r;
    green = ball_g;
    blue  = ball_b;
end
```

① Object-mapped

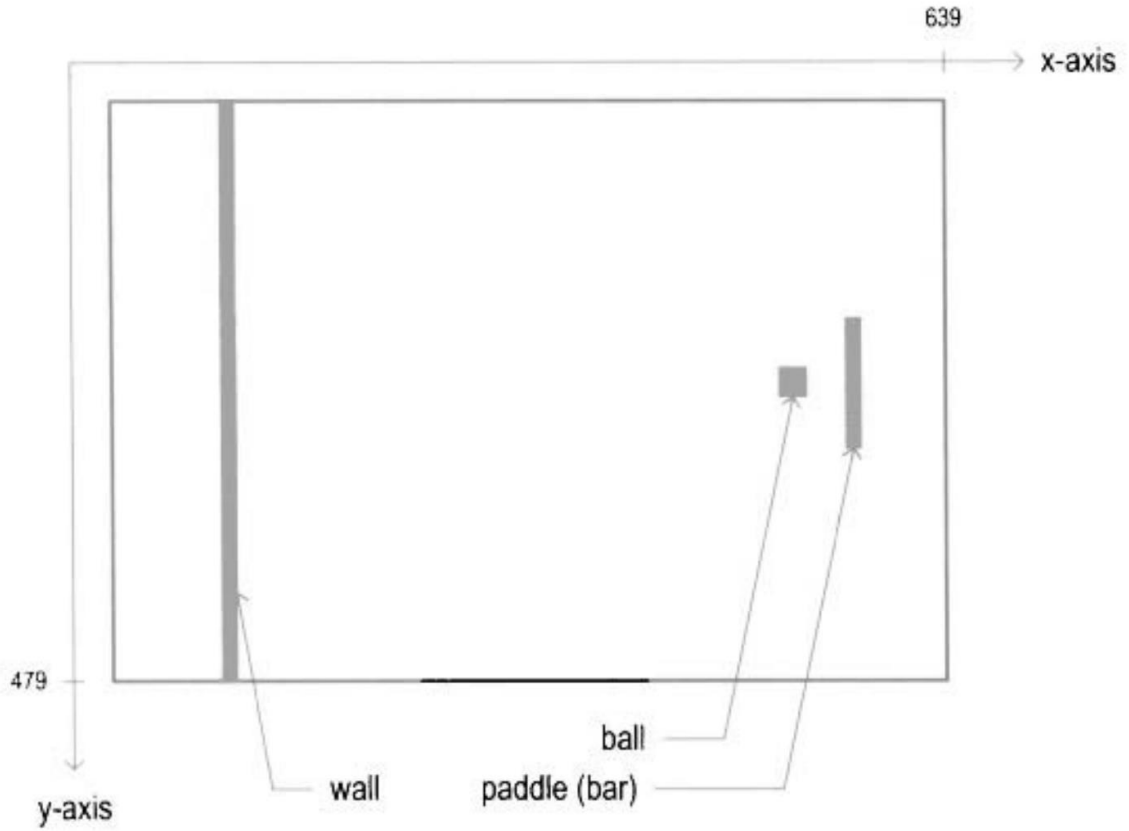
乒乓球：代码 静止

```
module pingpong_screen_still( input  logic video0n,
    input  logic [10:0] pix_x, pix_y, //扫描屏幕坐标
    output logic [3:0] red, green, blue );

    localparam
      MAX_X = 640,  MAX_Y = 480,  //(0,0) to (639,479)
      // 屏幕参数
      H_SYNC    =  96, // horizontal sync width
      H_BACK    =  48, // left border (back porch)
      H_SYNC_START = H_SYNC + H_BACK, //行显示后沿 = 144(96+48)
      V_SYNC    =   2, // vertical sync lines
      V_TOP     =  29, // vertical top border
      V_SYNC_START = V_SYNC + V_TOP, //场显示后沿 =  31(2+29)
      // === wall ===
      WALL_X_L = H_SYNC_START + 30, // left boundary
      WALL_X_R = H_SYNC_START + 40, // right boundary
      // === bar ===
      BAR_X_L = H_SYNC_START + 600, // left boundary
      BAR_X_R = H_SYNC_START + 605, // right boundary
      BAR_Y_SIZE = 70,              // bar 的高度
      BAR_Y_T = V_SYNC_START + MAX_Y/2 - BAR_Y_SIZE/2, //Top=204
      BAR_Y_B = BAR_Y_T + BAR_Y_SIZE -1, // bottom boundary
      // === ball ===
      BALL_SIZE = 8,
      BALL_X_L = H_SYNC_START + 580,       // left boundary
      BALL_X_R = BALL_X_L + BALL_SIZE - 1, // right boundary
      BALL_Y_T = V_SYNC_START + MAX_Y/2 -BALL_SIZE/2,//top boundary
      BALL_Y_B = BALL_Y_T + BALL_SIZE - 1;       // bottom boundary

    logic wall0n, bar0n, ball0n;
    logic [3:0] wall_r, wall_g, wall_b;
    logic [3:0] bar_r,  bar_g,  bar_b;
    logic [3:0] ball_r, ball_g, ball_b;

    // (wall) : pixel within wall
    assign wall0n = (WALL_X_L <= pix_x) && (pix_x <= WALL_X_R);
    assign wall_r = 4'b0000;
    assign wall_g = 4'b0000;
    assign wall_b = 4'b1111; // blue

    // (bar) :  pixel within bar
    assign bar0n = ((BAR_X_L<=pix_x) && (pix_x<=BAR_X_R) &&
                    (BAR_Y_T<=pix_y) && (pix_y<=BAR_Y_B));
    assign bar_r = 4'b0000;
    assign bar_g = 4'b1111; // green
    assign bar_b = 4'b0000;

    // (ball): pixel within squared ball
    assign ball0n = ((BALL_X_L<=pix_x) && (pix_x<=BALL_X_R) &&
                     (BALL_Y_T<=pix_y) && (pix_y<=BALL_Y_B));
    assign ball_r = 4'b1111; // red
    assign ball_g = 4'b0000;
    assign ball_b = 4'b0000;

    always_comb   // ===== rgb 输出 ==
        if (~video0n)
        begin
            red   = 4'b0000; // blank
            green = 4'b0000; // blank
            blue  = 4'b0000; // blank
        end
        else if (wall0n)
        begin
            red   = wall_r;
            green = wall_g;
            blue  = wall_b;
        end
        else if (ball0n)
        begin
            red   = ball_r;
            green = ball_g;
            blue  = ball_b;
        end
        else if (bar0n)
        begin
            red   = bar_r;
            green = bar_g;
            blue  = bar_b;
        end
        else
        begin // gray background
            red   = 4'b1110;
            green = 4'b1110;
            blue  = 4'b1110;
        end
endmodule
```

乒乓球：顶层代码

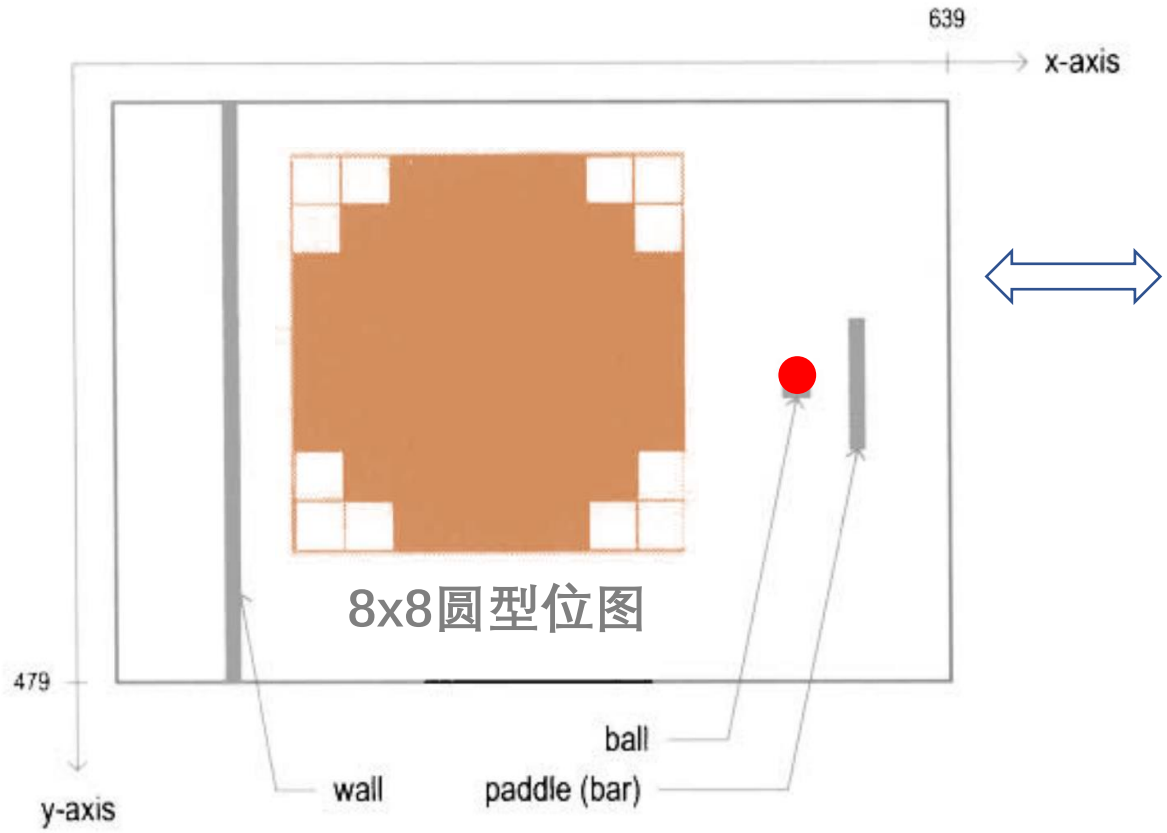
静止

```
1  module pingpog_screen_still_Top(
2      input  logic CLK100MHZ, BTND,   //为了操作方便
3      output logic VGA_HS, VGA_VS,
4      output logic [3:0] VGA_R, VGA_G, VGA_B);
5
6      logic clk25MHz, videoOn;
7      logic [10:0] pixel_x, pixel_y;
8
9      clkDiv C1(.clk(CLK100MHZ), .clr(BTND),
10               .clk25MHz(clk25MHz));
11
12     VGA640x480 V1(.clk(clk25MHz), .clr(BTND),   // Input
13         .hSync(VGA_HS),    .vSync(VGA_VS),      // Output ***
14         .xPixel(pixel_x), .yPixel(pixel_y),     // Output
15         .displayOn(videoOn));                   // Output
16
17     pingpong_screen_still P1( .videoOn(videoOn),
18                               .pix_x(pixel_x),
19                               .pix_y(pixel_y),
20                               .red(VGA_R),      // Output ***
21                               .green(VGA_G),    // Output ***
22                               .blue(VGA_B));    // Output ***
23 endmodule
```



乒乓球：ball（圆球）

静止




```
// === ball ===
BALL_SIZE = 8,
BALL_X_L = H_SYNC_START + 580,        // left boundary
BALL_X_R = BALL_X_L + BALL_SIZE - 1, // right boundary
BALL_Y_T = V_SYNC_START + MAX_Y/2 -BALL_SIZE/2,//top boundary
BALL_Y_B = BALL_Y_T + BALL_SIZE - 1;        // bottom boundary
```

```
logic [2:0] rom_addr, rom_col; // ROM中8行、8列
logic rom_bit;                 // ROM中每个像素值
logic [7:0] rom_data;

always_comb        // image ROM
    case (rom_addr)
        3'h0: rom_data = 8'b0011_1100; //    ****
        3'h1: rom_data = 8'b0111_1110; //   ******
        3'h2: rom_data = 8'b1111_1111; //  ********
        3'h3: rom_data = 8'b1111_1111; //  ********
        3'h4: rom_data = 8'b1111_1111; //  ********
        3'h5: rom_data = 8'b1111_1111; //  ********
        3'h6: rom_data = 8'b0111_1110; //   ******
        3'h7: rom_data = 8'b0011_1100; //    ****
    endcase
```

```
assign rom_col  = pix_x[2:0] - BALL_X_L[2:0]; // ROM列
assign rom_addr = pix_y[2:0] - BALL_Y_T[2:0]; // ROM行
assign rom_bit  = rom_data[rom_col]; //ROMaddr行中col列值
```

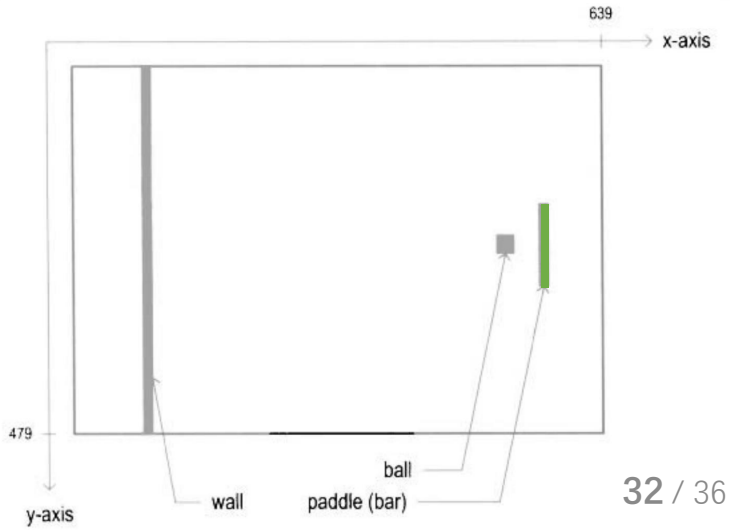
圆球一行中某一列的像素值

```
// pixel within ball
assign ballOn = ((BALL_X_L<=pix_x) && (pix_x<=BALL_X_R) &&
                 (BALL_Y_T<=pix_y) && (pix_y<=BALL_Y_B)) &
                rom_bit;   // round ball
assign ball_r = 4'b1111; // red
assign ball_g = 4'b0000; // red
assign ball_b = 4'b0000; // red
```

乒乓球：bar （球拍）运动

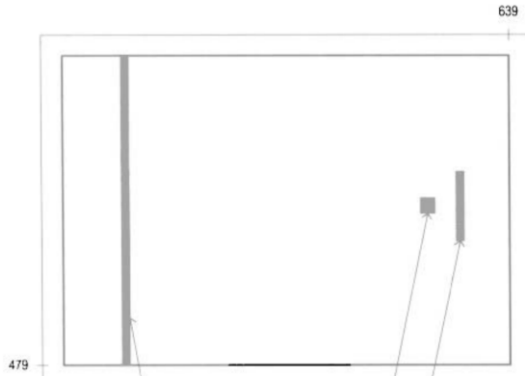
```
module pingpong_screen_move(
    input  logic clk, reset,      //clk: 屏幕60Hz刷新频率
    input  logic btnUp, btnDown, //控制Bar上下移动

logic [10:0] bar_y_t, bar_y_b; // bar top, bottom boundary

// new bar y-position
always @(posedge clk, posedge reset)
begin
    if (reset)
    begin
        bar_y_t <= BAR_Y_T;
    end
    else if (btnDown & (bar_y_b <= (V_SYNC_START+MAX_Y-1-BAR_V)))
        bar_y_t <= bar_y_t + BAR_V; // move down
    else if (btnUp & (bar_y_t >= (V_SYNC_START+BAR_V)))
        bar_y_t <= bar_y_t - BAR_V; // move up
end
assign bar_y_b = bar_y_t + BAR_Y_SIZE - 1;
```

```
assign barOn = ((BAR_X_L<=pix_x) && (pix_x<=BAR_X_R) &&
                (bar_y_t<=pix_y) && (pix_y<=bar_y_b)); //变量
assign bar_r = 4'b0000;
assign bar_g = 4'b1111; // green
assign bar_b = 4'b0000;
```



乒乓球：ball 运动

```
BALL_V = 2;     // 每次移动 ball 的距离
logic [10:0] ball_x_l, ball_x_r; // ball left, right boundary
logic [10:0] ball_y_t, ball_y_b; // ball top, bottom boundary
logic [10:0] ball_dx , ball_dy ; // ball x, y 增量

assign ball_x_r = ball_x_l + BALL_SIZE - 1;
assign ball_y_b = ball_y_t + BALL_SIZE - 1;

// pixel within ball
assign ballOn = ((ball_x_l<=pix_x) && (pix_x<=ball_x_r) &&  //变量
                 (ball_y_t<=pix_y) && (pix_y<=ball_y_b)) &  //变量
                 rom_bit;   // round ball
assign ball_r = 4'b1111; // red
assign ball_g = 4'b0000; // red
assign ball_b = 4'b0000; // red
```



别忘记修改ROM中的球位置

```
assign rom_col  = pix_x[2:0] - ball_x_l[2:0]; // ROM列
assign rom_addr = pix_y[2:0] - ball_y_t[2:0]; // ROM行
```

```
// new ball x,y-position
always @(posedge clk, posedge reset)    begin
    if (reset)   begin
        ball_x_l <= BALL_X_L;
        ball_y_t <= BALL_Y_T;
        ball_dx  <= -1 * BALL_V;
        ball_dy  <= -1 * BALL_V;
    end
    else   begin
        ball_x_l <= ball_x_l + ball_dx;
        ball_y_t <= ball_y_t + ball_dy;
        // ------------------ ball bounce back ----------------------
        if (ball_y_t <= V_SYNC_START)              //reach top screen
            ball_dy <= BALL_V;
        else if (ball_y_b >= V_SYNC_START + MAX_Y) //reach bottom screen
            ball_dy <= -1 * BALL_V;
        else if (ball_x_l <= WALL_X_R)             //reach left wall
            ball_dx <= BALL_V;
        else if ((ball_x_r >= BAR_X_L) &&          //reach right bar
                 (ball_x_r <= BAR_X_R) &&
                 (ball_y_b >= bar_y_t) &&  (ball_y_t <= bar_y_b))
            ball_dx <= -1 * BALL_V;
        else if (ball_x_r >= H_SYNC_START + MAX_X) //reach right screen
            ball_dx <= -1 * BALL_V;
    end
end
```

乒乓球：顶层代码

运动

```
1  module pingpong_screen_move_Top(
2      input  logic CLK100MHZ, BTND,   //为了操作方便
3      input  logic BTNU, BTNC,        //控制Bar上下
4      output logic VGA_HS, VGA_VS,
5      output logic [3:0] VGA_R, VGA_G, VGA_B);
6
7      logic clk25MHz, clk60Hz, videoOn;
8      logic [10:0] pixel_x, pixel_y;
9
10     clkDiv C1(.clk(CLK100MHZ), .clr(BTND),
11               .clk25MHz(clk25MHz));
12
13     VGA640x480 V1(.clk(clk25MHz), .clr(BTND),   // Input
14         .hSync(VGA_HS),    .vSync(VGA_VS),      // Output ***
15         .xPixel(pixel_x), .yPixel(pixel_y),     // Output
16         .displayOn(videoOn));                   // Output
17
18     clk60Hz C2(.clk(CLK100MHZ), .clr(BTND), .clk60Hz(clk60Hz));
19
20     pingpong_screen_move P1(.clk(clk60Hz),   .reset(BTND),
21                             .btnUp(BTNU),    .btnDown(BTNC),
22                             .videoOn(videoOn),
23                             .pix_x(pixel_x), .pix_y(pixel_y),
24                             .red(VGA_R),     // Output ***
25                             .green(VGA_G),   // Output ***
26                             .blue(VGA_B));   // Output ***
27 endmodule
```

【作业2】越狱游戏

左侧的墙被多层砖块替代，
当小球撞击到砖块时，
小球被弹回，
同时被撞击的砖块消失。

参考资料

Nexys4-DDR_Reference Manual.pdf