同济大学计算机系

数字逻辑课程综合实验报告

学　　号 _____

姓　　名 _____

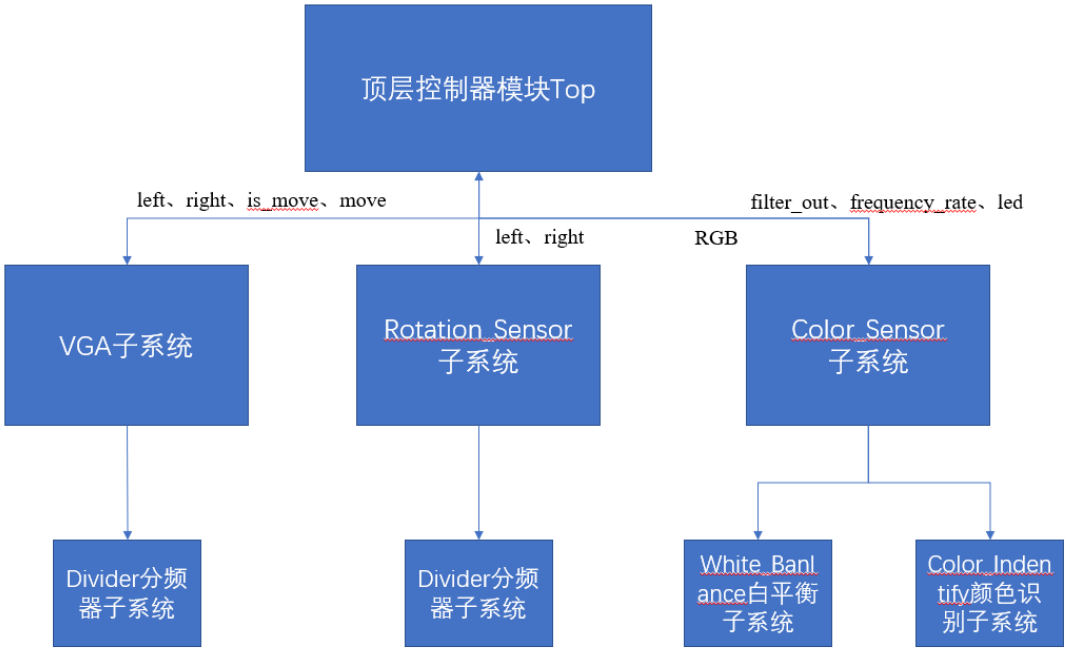
专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

基于 TCS3200 颜色传感器、旋转编码器和 VGA 的 FPGA 木板弹球小游戏

二、木板弹球数字系统总框图



① 首先底层进行分频器模块的实现，将原有 100MHz 的时钟进行任意整数倍分频，以满足不同模块不同信号的要求。

② 整体顶层控制器模块 Top，负责实例化各个模块和状态的转换，为 VGA 模块提供输出信号，为 Rotation_Sensor 模块提供控制信号并接受来自旋转编码器的顺逆时针的输出信号，为 Color_Sensor 模块提供控制信号并接受来自颜色传感器的颜色 RGB 值信号。

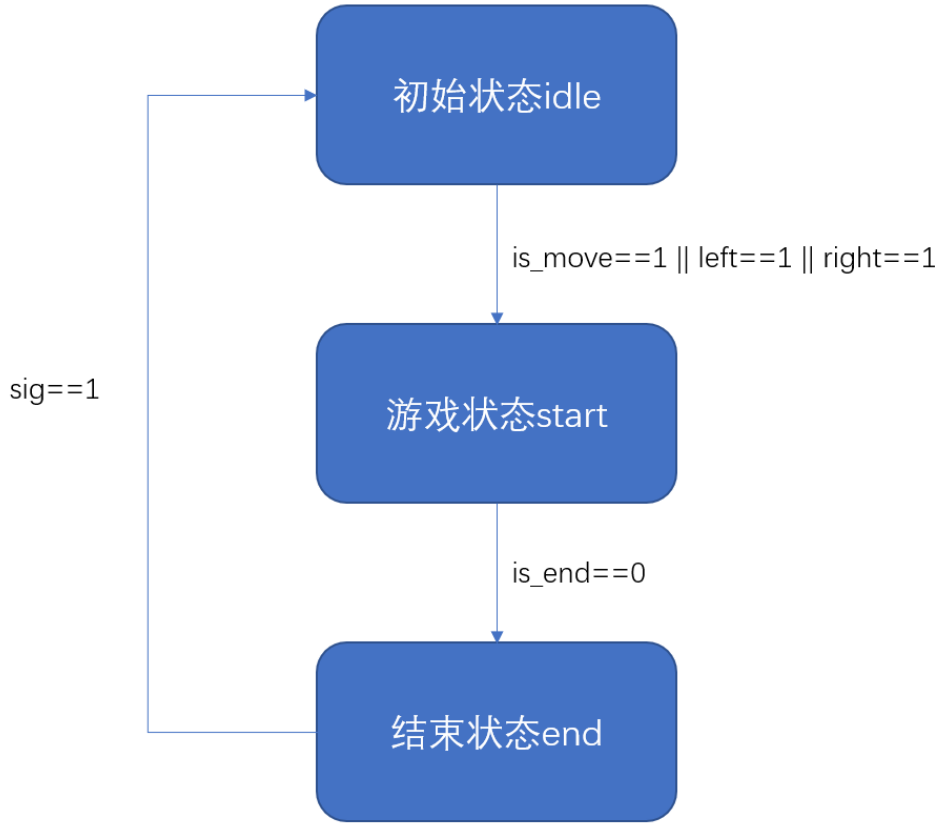
③ VGA 系统，显示基本图形界面，并在内部实现木板和小球的描绘和移动的函数表述，通过接受由 Top 模块传来的 left、right、is_move、move 的信号，对显示内容进行修改操作，使得木板得以顺逆时针旋转并且上下左右移动，小球可以进行正确的反弹。在游戏结束将 is_end 信号返回给 Top 模块。

④ Rotation_Sensor 旋转编码器系统，通过接收来自旋转编码器发出的 sa，sb 和 sw 信号的波形并对其进行分析，得到编码器是逆时针还是顺时针旋转的信号 left 和 right，并将其传回 Top 模块。

⑤ Color_Sensor 颜色传感器系统，通过接收 Top 模块的控制信号 filter_out、frequency_rate 和 led 信号对颜色传感器进行控制，颜色传感器返回每种颜色的 RGB 波形，依据频率进行解析后得到每种颜色的 RGB 值并返回 Top 模块。

三、系统控制器设计

系统 ASM 流程图



状态转移真值表

	现态		次态		转移条件		
	state		state		sig	is_end	move
idle	0	0	0	1	x	x	1
			0	0	x	x	0
start	0	1	1	0	x	1	x
			0	1	x	0	x
stop	1	0	0	0	1	x	x
			1	0	0	x	x

顶层控制器模块内容

```
module Top(
input clk,
input sig,
input sw,//复位信号，为 0 时复位
input sa,//左
input sb,//右
input frequency,
output    hs,
output    vs,
output    [3:0] Red,
output    [3:0] Green,
output    [3:0] Blue,
output left_test,
output right_test,
output [1:0] filter_out,
output led,
output [1:0] frequency_rate
      );



      wire left;//逆时针，对应向左
      wire right;//顺时针    对应向右

      wire is_end;

      reg signed [15:0]sin_alpha;
      reg signed [15:0]cos_alpha;

      wire rst=0;

      assign left_test=left;
      assign right_test=right;

      wire is_move;
      wire [1:0]move;

       color_top
Color(clk,frequency,is_move,move,filter
_out,led,filter_rate);
       Rotation_Sensor
rotate(clk,sa,sb,sw,rst,left,right);
       VGA_Module
vga(clk,left,right,is_move,move,sig,hs,v
s,Red,Green,Blue,is_end);

       reg [1:0]state;
       parameter
idle=2'b00,start=2'b01,stop=2'b10;
       always@(posedge clk)
       begin

       case (state)
       idle:begin
           if(is_move ||left||right)
           state=start;
           else
           state=idle;
           end
       start:begin
           if(is_end==1)
           state=stop;
           else
           state=start;
           end
       stop:begin
           if(sig==1)
           state=idle;
           else
           state=stop;
           end
       endcase

       end

endmodule
```

四、子系统模块建模

（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模的 verilog 代码）

1.分频器 Divider 模块

（1）描述

在 FPGA 中自带时钟频率 100MHz，有时对于模块的操作来说频率过于高，因此，需要使用分频器模块对 100MHz 进行分频使用，以满足实际需求。

在分频器实现过程，即设置一个 parameter 型参数 num，一个 reg 型计数器，每个时钟信号上升沿到来时对计数器进行加一操作，当加到参数 num 的一半时输出反向，如此往复，即可得到对应频率的新时钟波形。

（2）接口信号图

其中 clk 接 FPGA 自带时钟，num 为要进行的分频倍数，rst 是复位信号，输出 new_clk 为所需时钟

（3）Verilog 代码

```
module Divider(
    input I_CLK,
    input rst,
    output reg O_CLK
    );

    parameter num=20;
    reg [32:0]half=num/2;
    reg [32:0]count=0;
    initial
    begin
    if(count==0)
    O_CLK=0;
    end
always @(posedge I_CLK)
begin
if(rst==1)
begin
O_CLK=0;
count=0;
end
else
begin

count=count+1;
if(count==num)
count=0;
if(count>=0&&count<half)
O_CLK=0;
else //if(count>=half&&count<num)
O_CLK=1;
//else;
```

end

end
endmodule

2.旋转编码器 Rotation_Sensor 模块.

（1）描述

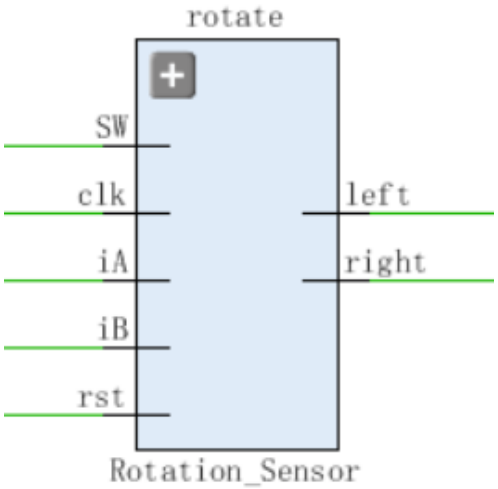
增量编码器是一种将旋转位移转换为一连串数字脉冲信号的旋转式传感器。通过旋转可以计数正方向和反方向转动过程中输出脉冲的次数，旋转计数不像电位计，这种转动计数是没有限制的。配合旋转编码器上的按键，可以复位到初始状态，即从 0 开始计数。

旋转编码器根据不同的旋转方向输出不同的严格时序信号。当旋转编码器中心按键按下时，输出信号中 sw 变为 0，标志此时复位；当旋转编码器逆时针旋转时，sa 比 sb 优先到达高电平，即 sa 为上升沿时 sb 为低电平，之后 sb 也上升到高电平，并保持两个高电平的状态；当旋转编码器顺时针旋转时，sb 比 sa 优先到达高电平，即 sb 为上升沿时 sa 为低电平，之后 sa 也上升到高电平，并保持两个高电平的状态。

（2）旋转编码器接口定义

引脚号	标识	描述
1	SIA	与 SIB 组合判断编码器的旋转方向
2	SIB	与 SIA 组合判断编码器的旋转方向
3	SW	读取编码器的按键是否按下
4	GND	电源地
5	VCC	电源正(3.0V-5.3V)

（3）接口信号图



（4）调试说明

旋转编码器是旋钮型的器件，因此在使用中会产生因不同方向转动而导致的抖动影响，因此在实际使用中应使用软件消抖，将旋转编码器的抖动降低到最低。

在本程序中，使用分频器构造 2kHz 的时钟信号，并在 2kHz 时钟上升沿对 sa、sb、sw 信号进行延时锁存，即不是每时每刻都进行及时的保存，而是过一段时间之后再进行保存。之后用 A_state 和 B_state 两个状态量对旋转编码器的 sa 和 sb 信号进行高电平检测，使用 A_pos = (!A_state_reg) && A_state，A_neg = A_state_reg && (!A_state)信号检测 A 的上升沿和下降沿，原理是当 A 现在状态时高电平而锁存的状态时低电平时，代表此时 A 进行了由低到高的电平转换，因此此时 A 为上升沿，下降沿同理。

使用 A_pos 和 A_neg 来表示上升沿和下降沿而不是 posedge，negedge 来表示，是因为在转变旋转方向时可能会发生信号的抖动，使得 A 信号不正常的上升下降，而旋转编码器判断旋转方向主要看上升沿和下降沿时与另一个信号的组合，因此会产生不正确的旋转信号。而使用 A_pos 或 A_neg 后由于延时锁存，便可以消除在延时锁存期间产生的不稳定信号，从而得到更为精准的结果，实现防抖动。

（5）Verilog 代码如下

```
//旋转编码器模块，通过记录编码器向左旋转的时间来给出角度
module Rotation_Sensor(//旋转编码器
    input clk,              //系统时钟
    input iA,               //旋转编码器 A 管脚
    input iB,               //旋转编码器 B 管脚
    input SW,               //旋转编码器 D 管脚
    input rst,              // 编 码 器 复 位，高有效
    output left,
    output right
);

    reg [1:0] odata; //输出编码

    integer i=0;
//计数器周期为 500us，控制键值采样频率
    wire clk_2kHZ;
    Divider #50000 divide(clk,rst,clk_2kHZ);//分频器，分成 2kHz

    //设置暂存状态
    reg  key_a_r;
    reg key_b_r;
    reg key_sw_r;

//针对 A、B、SW 管脚分别做简单去抖操作，
//在每个采样时间先记录下当前的旋转状态，并给到暂存变量，之后在

//对旋转编码器的输入缓存，消除亚稳态同时延时锁存
    always@(posedge clk_2kHZ) begin
        i=i+1;
        key_a_r     <= iA;//进行暂存操作
        key_b_r     <= iB;
        if(i==40) begin  //对于按键信号采用 20ms 周期采样的方法，40*500us = 20ms
            i=0;
            key_sw_r <= SW;//进行暂
```

```
存操作
        end
        else
            key_sw_r <=     key_sw_r;
    end

    reg     key_sw_r1;// 延 时 锁 存 变
量

    //对按键 D 信号进行延时锁存
    always@(posedge clk_2kHZ)
        key_sw_r1 <= key_sw_r;

    wire    A_state     = key_a_r &&
iA; //旋转编码器 A 信号高电平状态检
测
    wire    B_state     = key_b_r &&
iB; //旋转编码器 B 信号高电平状态检
测

    reg   A_state_reg;

    //延时锁存
    always@(posedge clk_2kHZ)
        A_state_reg <= A_state;

    //旋转编码器 A 信号的上升沿和下
降沿检测
    wire    A_pos   =   (!A_state_reg)
&& A_state;
    wire    A_neg   = A_state_reg &&
(!A_state);

    //通过旋转编码器 A 信号的边沿和
B 信号的电平状态的组合判断旋转编
码器的操作，并输出对应的脉冲信号
    always@(posedge clk_2kHZ or
posedge rst)begin
        if(rst==1) odata=2'b00;
        else if(A_pos && !B_state)
odata=2'b01;//正向旋转   逆时针
        else if(A_neg && B_state)
odata=2'b00;
        else if(A_pos && B_state)
odata=2'b10;
        else if(A_neg && !B_state)
odata=2'b00;     //反向旋转 顺时针
        else    if(key_sw_r1    &&
(!key_sw_r)) odata=2'b11;       //  旋
转 编 码 器  D  信 号 下 降 沿 检 测
odata=2'b00;
        else if ((!key_sw_r1) &&
key_sw_r)odata=2'b00;  //D 信号上升
沿将其置零
        else;
    end

    assign left=odata[0];
    assign right=odata[1];

endmodule
```

3.颜色传感器 Color_Sensor 模块

（1）器件描述

TCS3200 颜色传感器是一款全彩的颜色检测器，包括了一块 TAOS TCS3200RGB 感应芯片和 4 个白光 LED 灯，TCS3200 能在一定的范围内检测和测量几乎所有的可见光。它适合于色度计测量应用领域。比如彩色打印、医疗诊断、计算机彩色监视器校准以及油漆、纺织品、化妆品和印刷材料的过程控制。

通常所看到的物体颜色，实际上是物体表面吸收了照射到它上面的白光(日光)中的一部分有色成分，而反射出的另一部分有色光在人眼中的反应。白色是由各种频率的可见光混合在一起构成的，也就是说白光中包含着各种颜色的色光(如红 R、黄 Y、绿 G、青 V、蓝 B、紫 P)。根据德国物理学家赫姆霍兹(Helinholtz)

的三原色理论可知，各种颜色是由不同比例的三原色(红、绿、蓝)混合而成的。

TCS3200D 传感器有红绿蓝和清除 4 种滤光器，可以通过其引脚 S2 和 S3 的高低电平来选择滤波器模式。

TCS3200D 有可编程的彩色光到电信号频率的转换器，当被测物体反射光的红、绿、蓝三色光线分别透过相应滤波器到达 TAOS TCS320ORGB 感应芯片时，其内置的振荡器会输出方波，方波频率与所感应的光强成比例关系，光线越强，内置的振荡器方波频率越高。TCS3200 传感器有一个 OUT 引脚，它输出信号的频率与内置振荡器的频率也成比例关系，它们的比率因子可以靠其引脚 S0 和 S1 的高低电平来选择。

在正式测试颜色之前需要进行白平衡的操作，来告诉颜色传感器什么样的颜色是白色。

白平衡校正方法是:把一个白色物体放置在 TCS3200 颜色传感器之下，两者相距 10mm 左右，点亮传感器上的 4 个白光 LED 灯，然后选通三原色的滤波器，让被测物体反射光中红、绿、蓝三色光分别通过滤波器，计算 1s 时间内三色光对应的 TCS3200 传感器 OUT 输出信号脉冲数（单位时间的脉冲数包含了输出信号的频率信息），再通过正比算式得到白色物体 RGB 值 255 与三色光脉冲数的比例因子。有了白平衡校正得到的 RGB 比例因子，则其它颜色物体反射光中红、绿、蓝三色光对应的 TCS3200 输出信号 1s 内脉冲数乘以 R、G、B 比例因子，即可换算出了被测物体的 RGB 标准值。

（2）颜色传感器接口定义

引脚号	标识	描述
1	LED	控制四个 LED 灯的状态
2	OUT	读取 RGB 三原色对应的输出频率
3	S3	与 S2 端口组合选择不同色光的滤波器
4	S2	与 S3 端口组合选择不同色光的滤波器
5	S1	与 S0 端口组合选择不同的输出比例因子
6	S0	与 S1 端口组合选择不同的输出比例因子
7	GND	电源地
8	VCC	电源正(2.7V-5.5V)

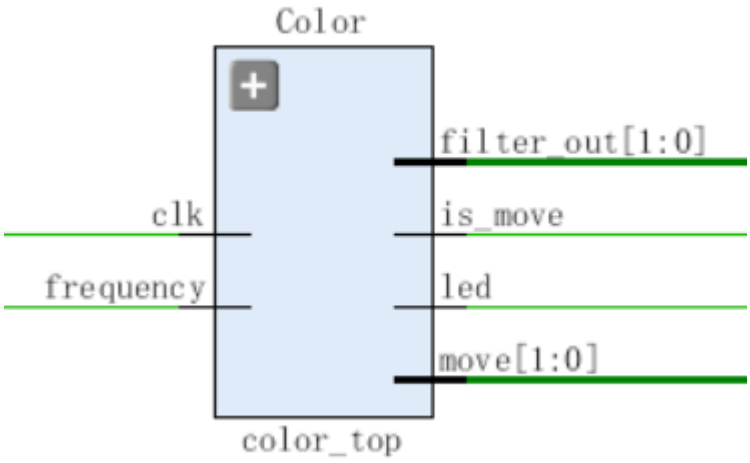
其中 S0 与 S1 组合选择比例因子

S0	S1	输出频率定标
L	L	关断电源
L	H	2%
H	L	20%
H	H	100%

S2 和 S3 组合选择滤波器

S2	S3	滤波器类型
L	L	红色
L	H	蓝色
H	L	无
H	H	绿色

（3）接口信号图



（4）调试说明

①在调试颜色传感器时，需要首先进行白平衡，白平衡过程的目的是告诉传感器与程序什么样的波形频率是白色的频率。所以在白平衡时，可以采用如下方法进行：首先设置 counter 变量用来记录 FPGA 自带时钟的上升沿次数，颜色传感器的滤波器选择按顺序选择红绿蓝色并接收输出波形，首先是红色，当红色对应的计数器计数至标准值（设为 255）时，记录在此间隔中时钟的上升沿个数，蓝色和绿色同理，由此得到，在红绿蓝色分别记录到 255 时的自带时钟上升沿的个数作为标准参数。

②颜色识别版块，因为在白平衡模块中得到了每个纯色的输出波形的固定周期对应的板子自带时钟周期，并且设置固定周期个数为 255，所以在识别颜色时，依次使用红绿蓝三色的滤波器，在该颜色对应的参数固定的时间内测量颜色输出波形的周期次数，得到的周期数即为该颜色的 RGB 数值，便可输出。

Verilog 代码如下

白平衡模块：

```
module White_Balance(
    input clk,          //时钟
    input  white_frequency, //颜色传感器输出的频率
    output reg [63:0] para_red,    //红色对应的参数
```

```
    output reg [63:0] para_green,   //绿色对应的参数
    output reg [63:0] para_blue,   //蓝色对应的参数
    output reg [1:0] filter_select,   //对于滤波器的选择
    output ready
    );

    reg [63:0]count=0;
    reg [63:0]red_count=0;
    reg [63:0]green_count=0;
    reg [63:0]blue_count=0;
    reg  [63:0]init_count=0;     //记录开始时计数器的个数，确保准确
    parameter standard_num=255;

    //使用固定红绿蓝周期个数，求出对应的标准时钟个数作为参数

    //记录时钟周期
    always @(posedge clk && !ready)
    begin
        count=count+1;
    end

    always @(posedge white_frequency && !ready)
    begin

        //if(red_count==0)
          // init_count=count;

        //求红色
        if(red_count<standard_num)
        begin
            red_count=red_count+1;
            filter_select=2'b00;
        end
        else if(red_count==standard_num)
        begin
            red_count=red_count+1;
para_red=count/*-init_count*/;
            filter_select=2'b11;      //转到绿色
        end
        else
        ;

        //求绿色

if(green_count<standard_num && red_count>standard_num)
        begin

green_count=green_count+1;
            filter_select=2'b11;
        end
        else if(green_count==standard_num)
        begin

green_count=green_count+1;

para_green=count-para_red/*-init_count*/;
            filter_select=2'b10;      //转到蓝色
        end
        else
        ;

        //求蓝色
        if(blue_count<standard_num && red_count>standard_num && green_count>standard_num)
        begin

blue_count=blue_count+1;
            filter_select=2'b10;
        end
        else if(blue_count==standard_num)
        begin

blue_count=blue_count+1;
```

```
para_blue=count-para_red-para_green/*
-init_count*/;
            end
            else
            ;


    end
```

颜色识别模块：

```
module Color_Identify(
    input clk,
    input frequency,
    input ready,
    input [63:0] red_para,
    input [63:0] green_para,
    input [63:0] blue_para,
    output reg [10:0] red,
    output reg [10:0] green,
    output reg [10:0] blue,
    output reg [1:0] filter,
    output ok
    );



    reg [63:0]count=0;
    reg [63:0]red_count=0;
    reg [63:0]green_count=0;
    reg [63:0]blue_count=0;
    reg  [63:0]init_count=0;     //记录
开始时计数器的个数，确保准确

    parameter standard_num=255;
    parameter  standard_value=255;//标
准颜色值
    //使用固定红绿蓝周期个数，求出
对应的标准时钟个数作为参数

    reg reset=0;//复位信号，每次测完
一次重新循环

    assign
ready=(red_count>standard_num)&&(gr
een_count>standard_num)&&(blue_cou
nt>standard_num);


endmodule
```

```
    //记录时钟周期
    always @(posedge clk && ready)
    begin
    if(!reset)
        count=count+1;
    else
       count=0;
    end

    always @(posedge frequency &&
ready)
    begin

    if(count==0)
    begin
        red_count=0;
        green_count=0;
        blue_count=0;
        reset=0;
   end

   if(count<red_para)
   begin
        filter=2'b00;
        red_count=red_count+1;
   end
   else     if(count>=red_para     &&
count<red_para+green_para)
   begin
```

```
            filter=2'b11;
            green_count=green_count+1;
        end
        else  if(count>=red_para+green_para
&&
count<red_para+green_para+blue_para)
        begin
            filter=2'b10;
            blue_count=blue_count+1;
        end
        else
if(count>=red_para+green_para+blue_p
ara)
        begin
            red=red_count-1;
            green=green_count-1;
            blue=blue_count-1;
            reset=1;
        end
        else;

         end

        assign
ok=(count>=red_para+green_para+blue
_para)?1:0;

endmodule
```

4.VGA 显示模块

（1）模块描述

VGA（Video Graphics Array）即视频图形阵列，是 IBM 在 1987 年推出的使用模拟信号的一种视频传输标准，在当时具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。这个标准对于现今的个人电脑市场已经十分过时。即使如此，VGA 仍然是最多制造商所共同支持的一个标准，个人电脑在加载自己的特殊驱动程序之前，都必须支持 VGA 的标准。

（2）原理介绍

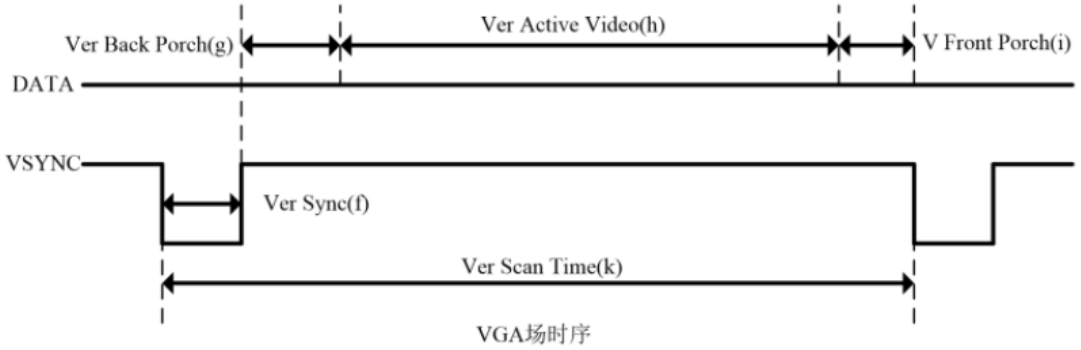
VGA 显示器扫描方式从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即屏幕的刷新频率，对于本次使用的为 60Hz

VGA 的时序主要包括行时序与场时序两个部分。

其中行时序主要包括：行同步(Hor Sync) 、行消隐(Hor Back Porch) 、行视频有效(Hor Active Video)和行前肩(Hor Front Porch)这四个参数，行时序的时序图如下图所示

VGA行时序

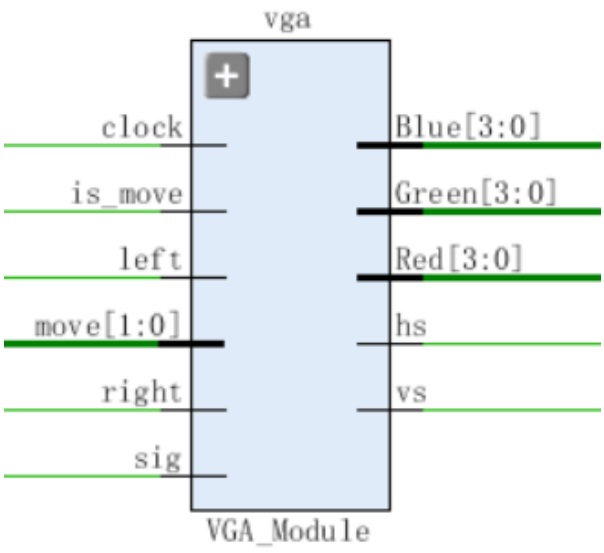
场时序主要包括：场同步(Ver Sync) 、场消隐(Ver Back Porch) 、场视频有效(Ver Active Video)和场前肩(Ver Front Porch)这四个参数，场时序的时序图如下图所示



本次使用的 VGA 的分辨率为 480*640，使用的时钟频率为 25.175MHz，因此各个参数为

```
parameter       H_SYNC_PULSE      =   96  ,
                H_BACK_PORCH      =   48  ,
                H_ACTIVE_TIME     =   640 ,
                H_FRONT_PORCH     =   16  ,
                H_LINE_PERIOD     =   800 ;

parameter       V_SYNC_PULSE      =   2   ,
                V_BACK_PORCH      =   33  ,
                V_ACTIVE_TIME     =   480 ,
                V_FRONT_PORCH     =   10  ,
                V_FRAME_PERIOD    =   525 ;
```

（3）功能框图



（4）Verilog 代码

因 VGA 模块中有较多的计算和控制部分，因此在这里省去这些代码，仅展示与 VGA 原理有关的部分代码。

```
module VGA_Module(
    input clock,
    input left,//逆时针
    input right,//顺时针
    input is_move,
    input [1:0]move,
    input sig,
    output reg hs,
    output reg vs,
    output reg [3:0] Red,
    output reg [3:0] Green,
    output reg [3:0] Blue,
    output reg is_end
    );
    // 分辨率为640*480时行时序各个参数定义
    parameter        H_SYNC_PULSE    =   96  ,
                     H_BACK_PORCH    =   48  ,

H_ACTIVE_TIME     =   640 ,

H_FRONT_PORCH     =   16  ,

H_LINE_PERIOD     =   800 ;

    // 分辨率为640*480时场时序各个参数定义
    parameter        V_SYNC_PULSE    =   2   ,
                     V_BACK_PORCH    =   33  ,

V_ACTIVE_TIME     =   480 ,
                     V_FRONT_PORCH   =   10  ,

V_FRAME_PERIOD    =   525 ;

    wire active_flag;//激活信号为1时才可以显示
   reg                     [11:0]
h_count=0         ; // 行时序计数器
   reg                     [11:0]
v_count=0         ; // 列时序计数器

   reg   clk_50M=0;
   reg    clk_25M =0;

   //产生50MHz频率
   always @(posedge clock)
   begin
        clk_50M=~clk_50M;
   end
   //产生25MHz的基准时钟
   always  @(posedge clk_50M)
   begin
        clk_25M=~clk_25M;
   end

    wire clk_100HZ;
    wire rst=0;
   Divider               #1000000
divide(clock,rst,clk_100HZ);// 分频器，分成100Hz

   //设置行计数器
   always @(posedge clk_25M)
   begin

       if(h_count<H_SYNC_PULSE)
       hs=0;
       else
       hs=1;

        if(h_count              ==
```

```
H_LINE_PERIOD - 1)
        h_count=0;
        else
        h_count=h_count+1;


   end


   //产生场时序
   always @(posedge clk_25M)
   begin
        if(v_count<V_SYNC_PULSE)
        vs=0;
        else
        vs=1;


if(v_count==V_FRAME_PERIOD-1)
        v_count=0;
        else
if(h_count==H_LINE_PERIOD - 1)
        v_count=v_count+1;
        else
        v_count=v_count;

   end

   //必须在 active 区间内才我那个屏幕上进行输出
   assign      active_flag      =
(h_count   >=   (H_SYNC_PULSE   +
H_BACK_PORCH                    ))
&&

(h_count   <=   (H_SYNC_PULSE   +
H_BACK_PORCH   +   H_ACTIVE_TIME))
&&

(v_count   >=   (V_SYNC_PULSE   +
V_BACK_PORCH                    ))
&&

(v_count   <=   (V_SYNC_PULSE   +
V_BACK_PORCH + V_ACTIVE_TIME))  ;
   //改动 RGB 的值，进行输出
   always @ (*)
   begin
   if(active_flag)
   begin
           //专门描绘边缘

if((v_count>Column_base         +
V_ACTIVE_TIME/10)               &&
(v_count<Column_base            +
9*V_ACTIVE_TIME/10)&&((h_count<L
ine_base  +  H_ACTIVE_TIME/10)  ||
(h_count>Line_base              +
9*H_ACTIVE_TIME/10              &&
(v_count<Column_base            +
5*V_ACTIVE_TIME/10              ||
v_count>Column_base             +
6*V_ACTIVE_TIME/10))))
           begin  //边缘蓝色（留一个空）

if(is_move==1)
                                begin

                Red             =
4'b0000    ;
                Green           =
4'b1111    ;
                Blue            =
4'b0000    ;
            end
            else
if((v_count<Column_base         +
V_ACTIVE_TIME/10)               ||
(v_count>Column_base            +
9*V_ACTIVE_TIME/10) )
                begin   //边缘蓝色
                    Red         =
4'b0000    ;
                    Green       =
4'b1111    ;
                    Blue        =
```

```
4'b0000    ;
                end

            else
            begin

                //先画两边的球
                dis=(y_center  -
v_count) * cos_alpha + (x_center -
h_count)* sin_alpha;
                dis2=(y_center  -
v_count) * sin_alpha - (x_center -
h_count)* cos_alpha;


if((h_count*float_transfer      -
x_cir1)*(h_count*float_transfer -
x_cir1) + (v_count*float_transfer
-                        y_cir1)
*(v_count*float_transfer - y_cir1)
<=          half_thick          *
half_thick*float_transfer*float_
transfer )
                begin
                    Red         =
4'b1111    ;
                    Green       =
4'b0000   ;
                    Blue        =
4'b0000    ;
                end
                else
if((h_count*float_transfer      -
x_cir2)*(h_count*float_transfer -
x_cir2) + (v_count*float_transfer
-                        y_cir2)
*(v_count*float_transfer - y_cir2)
<=          half_thick          *
half_thick*float_transfer*float_
transfer )
                begin
                    Red         =
4'b1111    ;
                    Green       =
4'b0000   ;
                    Blue        =
4'b0000    ;
                end

                else   if((dis
<=  half_thick*float_transfer  &&
dis>=
-half_thick*float_transfer)&&(  d
is2 >= -float_transfer*length/2 &&
dis2 <= float_transfer*length/2))
                begin

                    Red     =
4'b1111    ;
                    Green   =
4'b0000   ;
                    Blue    =
4'b0000    ;
                end
                //画球
              else  if((h_count
-  x_ball)*(h_count  -  x_ball)  +
(v_count   -   y_ball)*(v_count   -
y_ball)<=r_ball*r_ball/*1000000*
/)
                    begin

                        Red
=  4'b0000    ;

Green =  4'b1111   ;
                        Blue
=  4'b0000    ;
                    end

                //其他地方
              else
                    begin
                        Red
=  4'b0000    ; // 黑色
```

```
Green =  4'b0000    ;
                                Blue        end
=  4'b0000     ;
                            end             end
          end                               endmodule
```

五、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

仅保留了白平衡版块的 testbench 文件，其余的测试基本是直接进行合成 bit 流下板测试

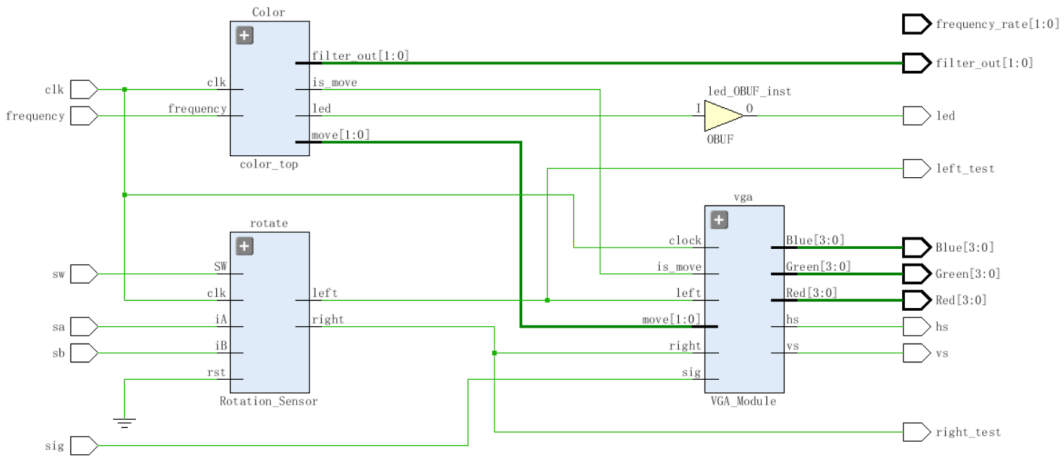
（1） VGA_try_tb

```
module VGA_try_tb;

    reg clock;
    wire hs;
    wire vs;
    wire [3:0] Red;
    wire [3:0] Green;
    wire [3:0] Blue;

    reg white_frequency; //颜色传感器输出的频率
    wire [63:0] para_red;   //红色对应的参数
    wire [63:0] para_green;   //绿色对应的参数
    wire [63:0] para_blue;   //蓝色对应的参数
    wire [1:0] filter_select;   //对于滤波器的选择

    VGA_Module uut(clock,hs,vs,Red,Green,Blue);
    //White_Balance
    WB(clock,white_frequency,para_red,para_green,para_blue,filter_select);

        always
        begin
            #0.25
            white_frequency=1;
            #0.25
            white_frequency=0;

        end

        always
        begin
        #0.125
        clock=1;
        #0.125
        clock=0;
        end


endmodule
```

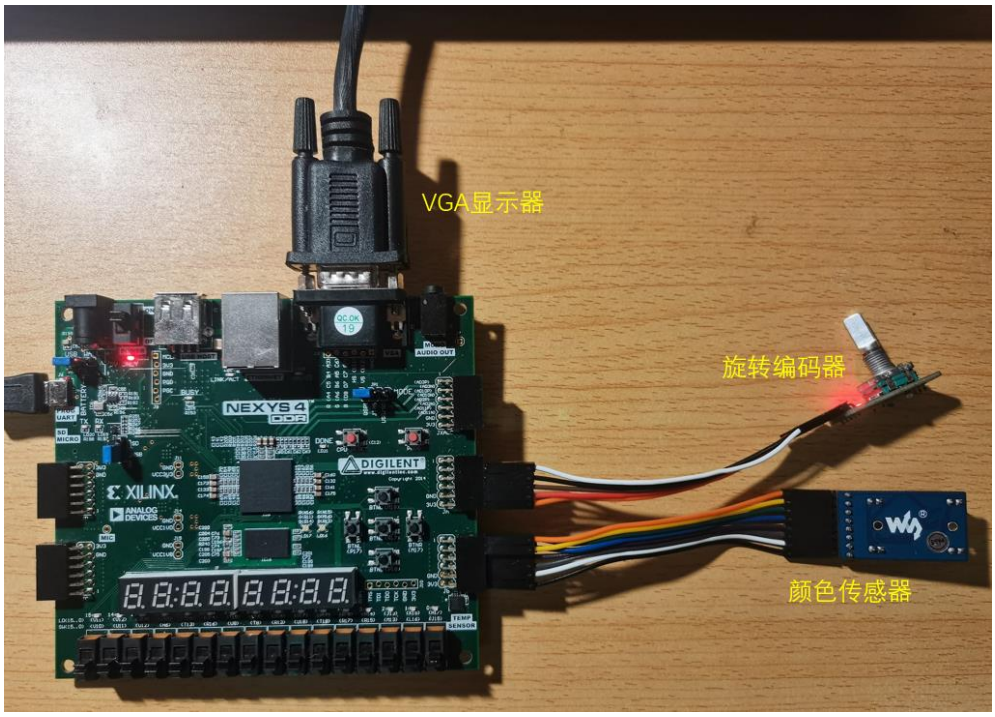
六、实验结果

（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

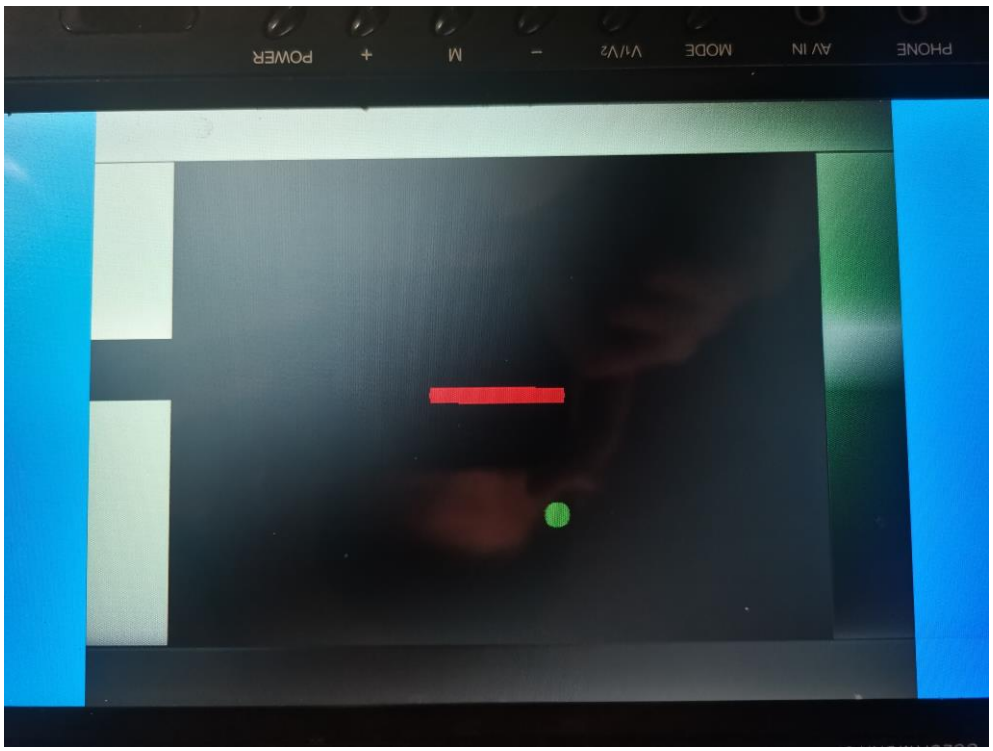
（1）Vivado 综合结果



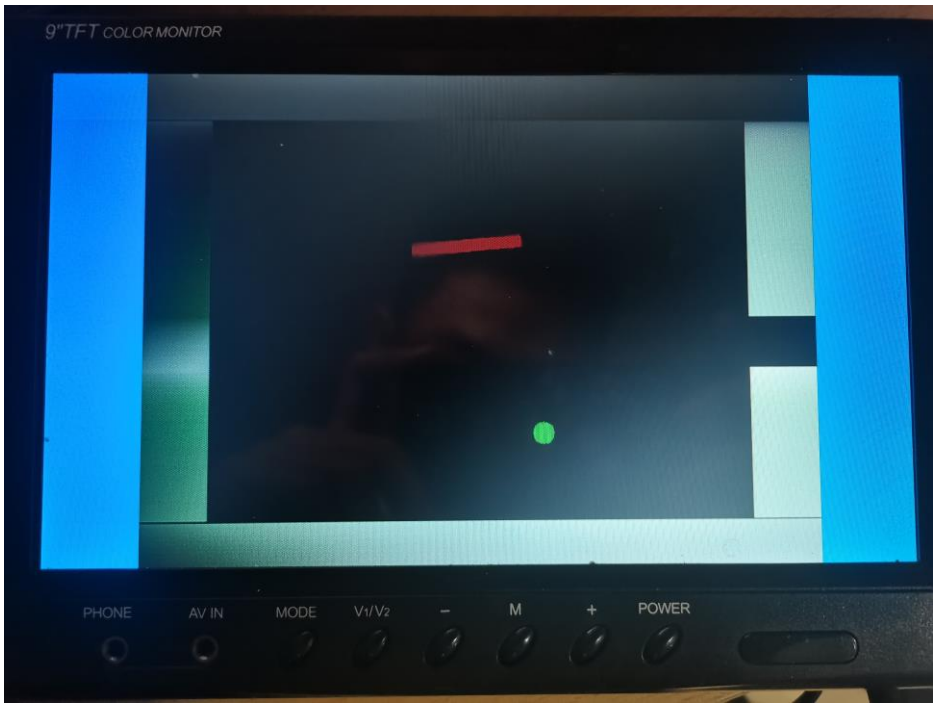
（2）开发板连接情况



（3）屏幕初始情况



（4）游戏过程中屏幕显示情况



（5）游戏成功结束时屏幕显示情况

此时为静态画面，当将开发板上的拨码开关 J15 拨开后，即可将屏幕解锁，继续游戏

七、结论

本数字系统由旋转编码器模块，颜色传感器模块，VGA 显示模块共同组成，由 Top 顶层模块进行整体的实例化与控制。

在每个模块的实现过程中都进行了不同方法的测试与调试以保证最后进行模块合并时可以正确合并。

旋转木板弹球小游戏初始木板不动，小球进行左右方向的来回反弹，之后可以借助旋转编码器实现对木板的顺逆时针旋转，借助颜色传感器实现对木板的上下左右移动，在反弹过程中小球与四周墙壁按照反射定律进行反弹，与木板是分离后按照反射定律进行反弹，当小球落到规定的空档位置的时候，游戏结束。若此时按动开发板上的拨码开关 J15，便可继续游戏。

八、心得体会及建议

① 在实现过程中主要的困难有三点，一是 VGA，旋转编码器和颜色传感器的配件的下板实现，第二是让木板可以顺逆时针顺畅旋转，第三是让小球实现在规定位置按照规定方向反弹。

② 旋转编码器的实现过程中主要问题是消除抖动的影响，因为旋钮型的信号反馈在顺逆时针的信号抖动，导致一开始并没有收到正常的信号，只后在网上搜索查找到延时锁存的消抖方法，大大提高了旋转编码器的稳定性，能对旋转信号做出正确的判断。但是目前旋转编码器还有一个问题是如何在停止旋转时将旋转信号归零，即做到停止旋转屏幕中的木板停止旋转，目前还没有实现。

③ 颜色传感器在实现过程中初始对于颜色的识别不很精准，在识别的过程

中总是红色较多，对于蓝色和绿色不很灵敏，导致一开始的识别颜色并不准确，之后调整后仅对较为明显的颜色进行识别，效果还比较好。

④ VGA 中主要的困难在于 sin 和 cos 的函数表示，利用泰勒函数展开以在 Verilog 中实现对 sin 和 cos 的计算，其中对于小数在 Verilog 中的表示，利用对于小数乘以 100 进行扩大表示，这样避免 Verilog 忽略小数取整，将精度近似在 0.01。另外在编写时拓展知道了 Verilog 中有符号数 signed 的使用，在进行图形化的过程中起了很大作用。第二个是小球反弹的问题，在初始编写代码的时候，小球在边界无法正确转向，后来分析程序发现应该是在边界时 theta 值改变，但是 cos 和 sin 还未改变，导致小球依然向墙内走入，在墙内时，theta 不断进行改变导致综合效果是不变，因此没有反弹的效果，在之后进行改变后可以正确实现，但是在小球和木板的碰撞过程还有一点 bug，只是在脱离后才反弹，应该是需要提前判断小球和木板的位置关系，进行处理，限于时间和精力，这部分先不做了。

做完整个项目之后再看做项目的过程，感觉一整个项目主要可以分为想法发现，前期调研，配件说明查询，配件实现和组合实现的过程，本次作业的想法是有的，并且通过这个想法去做出相应的解决方案，但是在配件的说明查询和实现的过程做的并不好，每次没有充分调研就开始实验，导致实验次数的增加并且还得不到正确的结果，因此，在固定配件的实现之前，可以参考配件说明书，配件的现有代码来实现他的基本功能。

另外，调试的过程也是给我一些启发，每次下板调试前需要确定好基本逻辑，不是仅仅针对错误进行修改，还要弄明白整体的逻辑过程，这样才不会无谓的浪费时间。

最后，通过这次的大作业我感觉才算是对硬件有了一定的入门，逐渐适应用硬件的方式去解读 Verilog 代码，去思考底层实现逻辑，并且对于硬件的信息传输有了更新更具体的认识，总体来说对于硬件思维的启发还是很大的。

（1）本课程收获：

通过本学期的数字逻辑课程的学习我觉得主要有以下几点收获

① 学习了硬件数字逻辑的基本的概念、应用与方法，了解掌握了基本的组合逻辑、时序逻辑的功能电路，对存储逻辑有了浅层的认识，对于数字系统的分析与设计明白了基本的步骤和方法。

② 学习和入门了硬件编程语言 Verilog，逐步适应了在编写代码时以硬件的方式去思考实现过程，在阅读分析代码时以硬件的方式去构想逻辑电路，而不是用软件的方式，懂得了用底层硬件逻辑去架构功能电路，我想这一思想的转变应该是最重要的。

③ 通过大作业使得动手能力得到了很大的提高，为系统去编写代码，将各个模块进行组合形成一个数字系统，了解信号的传输方式，明白硬件的底层逻辑。

（2）对本课程建议

经过一个学期的学习，我认为有以下几点建议。

① 建议更多的讲解 Verilog 语法及使用，本学期使用两节课来讲解 Verilog

感觉并不是很会使用 Verilog，后期报错也是不清楚原因。

② 对于 Vivado 平台的使用，希望可以做出一份详细一些的参考文档，我认为这样会更好上手以及入门，否则在使用中很多的功能是不会使用的。

（3）结合数字芯片国内外现状深入谈自己的认识和体会

目前我们课程使用的 FPGA 芯片是美国 DIGILENT 公司研发生产的，使用的开发平台 Vivado 是美国 Xilinx 公司研发的，所以我们所使用的主要装备基本都不是国产的，并且我们现在还没有能力进行研发生产代替，因此倘若中美之间展开科技竞争，美国对中国实行更加严厉的技术封锁，我们没有办法继续使用这些器件和工具，那么对我们的打击是巨大的。因此我们需要未雨绸缪，始终依靠自己，不能被他人卡脖子，我们现在使用美国的产品，我们可以在参考已有产品的特性，并且逐步了解内在的实现逻辑，最终打造出中国自己的产品，实现技术独立。

当然，我们也是这里面的一份子，我们不能仅仅考虑自身的个人需求，更需要为国家，为民族在世界舞台上挺直腰杆做出努力，不断探索硬件和数字逻辑中的知识，攻克其中的难关，为我国早日实现科技独立做出自己的贡献。

下面附上 VGA 模块的全部代码：

```
`timescale 1ns / 1ps
module VGA_Module(
    input clock,
    input left,//逆时针
    input right,//顺时针
    input is_move,
    input [1:0]move,
    input sig,
    output reg hs,
    output reg vs,
    output reg [3:0] Red,
    output reg [3:0] Green,
    output reg [3:0] Blue,
    output reg is_end
    );
    // 分辨率为 640*480 时行时序各个参数定义
    parameter       H_SYNC_PULSE      =   96  ,
                    H_BACK_PORCH      =   48  ,
                    H_ACTIVE_TIME     =  640 ,
                    H_FRONT_PORCH     =   16  ,
                    H_LINE_PERIOD     =  800 ;

    // 分辨率为 640*480 时场时序各个参数定义
    parameter       V_SYNC_PULSE      =   2   ,
                    V_BACK_PORCH      =   33  ,
```

```
                    V_ACTIVE_TIME       =    480 ,
                    V_FRONT_PORCH       =    10  ,
                    V_FRAME_PERIOD      =    525 ;
//实验，将屏幕分成 8 快
parameter           COLOR_BAR_WIDTH     =    H_ACTIVE_TIME / 8   ;




 wire active_flag;//激活信号     为 1 时才可以显示
reg [11:0]          h_count=0               ; //  行时序计数器
reg [11:0]          v_count=0               ; //  列时序计数器

reg     clk_50M=0;
reg      clk_25M =0;

//产生 50MHz 频率
always @(posedge clock)
begin
       clk_50M=~clk_50M;
end
//产生 25MHz 的基准时钟
always   @(posedge clk_50M)
begin
       clk_25M=~clk_25M;
end

 wire clk_100HZ;
 wire rst=0;
Divider #1000000 divide(clock,rst,clk_100HZ);//分频器，分成 100Hz

//设置行计数器
always @(posedge clk_25M)
begin

      if(h_count<H_SYNC_PULSE)
      hs=0;
      else
      hs=1;

       if(h_count == H_LINE_PERIOD - 1)
       h_count=0;
       else
       h_count=h_count+1;
```

```
    end


    //产生场时序
    always @(posedge clk_25M)
    begin
        if(v_count<V_SYNC_PULSE)
        vs=0;
        else
        vs=1;

        if(v_count==V_FRAME_PERIOD-1)
        v_count=0;
        else if(h_count==H_LINE_PERIOD - 1)
        v_count=v_count+1;
        else
        v_count=v_count;

    end

    //必须在 active 区间内才我那个屏幕上进行输出
    assign active_flag = (h_count >= (H_SYNC_PULSE + H_BACK_PORCH )) &&
                         (h_count <= (H_SYNC_PULSE + H_BACK_PORCH + H_ACTIVE_TIME)) &&
                         (v_count >= (V_SYNC_PULSE + V_BACK_PORCH )) &&
                         (v_count <= (V_SYNC_PULSE + V_BACK_PORCH + V_ACTIVE_TIME)) ;

    parameter Line_base=H_SYNC_PULSE+H_BACK_PORCH;
    parameter Column_base=V_SYNC_PULSE+V_BACK_PORCH;
    parameter half_thick = 6;
    parameter length = 100;
    parameter r_ball = 10;
    parameter speed_ball = 10;
    parameter speed_board = 40;
    parameter pi = 314;
    parameter float_transfer=100;


    reg [10:0]x_center = Line_base + H_ACTIVE_TIME/2;
```

```
    reg [10:0]y_center = Column_base + V_ACTIVE_TIME/2;
    reg [10:0]x_ball = Line_base + H_ACTIVE_TIME/2;
    reg [10:0]y_ball = Column_base + V_ACTIVE_TIME/2 - 60;
    reg signed [15:0]dis;
    reg signed [15:0]dis2;
    reg signed [15:0]alpha = 300;
    reg [15:0]temp_alpha;//用作中途计算之需
    reg signed [15:0]sin_alpha = 0;
    reg signed [15:0]cos_alpha = float_transfer;//float_transfer;//问题是现在这个位数太少，只有
6 位，最多 64，存不下来
    reg signed [15:0]theta = 0;
    reg [15:0]temp_theta = 0;//用作中途计算之需
    reg signed [15:0]sin_theta = 0;
    reg signed [15:0]cos_theta = float_transfer;//用来标记 cos(theta)的正负



    reg signed [15:0]dis_temp1;
    reg signed [15:0]dis_temp2;

    //修改角度 alpha 的值

    always @ (posedge clk_100HZ)
    begin

    if(is_end==1)
    alpha=alpha;

else
begin
    if(left==1 && right==0)
    alpha = alpha + 1;
    else if(left==0 && right==1)
    alpha = alpha - 1;
    else if(left==0 && right==0)
    alpha = alpha;
    else
    alpha=0;

    if(alpha>pi)
    alpha=alpha-pi;
    else if(alpha<0)
    alpha=alpha+pi;
    else;
```

```
end

    end


    //修改 sin 和 cos 的值
    //此处不能用 POSedge,,, 不是时钟，需要注意,,, 很隐蔽 的错误

    always@(alpha)
    begin

        //根据 alpha 和 theta 的位置给出相应的 sin 和 cos 正负 并给出其值
        if(alpha>=0&&alpha<pi/2)
        begin
        temp_alpha=alpha;

sin_alpha=((temp_alpha)-float_transfer*temp_alpha*temp_alpha*temp_alpha/(float_transfer*float_transfer*float_transfer)/6+float_transfer*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/120-float_transfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/5040);

cos_alpha=(float_transfer-float_transfer*(temp_alpha*temp_alpha)/(float_transfer*float_transfer)/2+float_transfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(float_transfer*float_transfer*float_transfer*float_transfer)/24-float_transfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/720);


        end
        else if(alpha>=pi/2 && alpha<pi)
        begin
        temp_alpha=alpha-pi/2;

sin_alpha=(float_transfer-float_transfer*(temp_alpha*temp_alpha)/(float_transfer*float_transfer)/2+float_transfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(float_transfer*float_transfer*float_transfer*float_transfer)/24-float_transfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/720);

cos_alpha=-((temp_alpha)-float_transfer*temp_alpha*temp_alpha*temp_alpha/(float_transfer*float_transfer*float_transfer)/6+float_transfer*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/120-float_tra
```

```
nsfer*(temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha*temp_alpha)/(
float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_tran
sfer)/5040);


        end

    end


//给出画板子的四个角的位置
        reg signed [15:0]x_left;
        reg signed [15:0]y_left;
        reg signed [15:0]x_right;
        reg signed [15:0]y_right;
        reg signed [15:0]x_cir1;
        reg signed [15:0]x_cir2;
        reg signed [15:0]y_cir1;
        reg signed [15:0]y_cir2;

        always@(*)
        begin
        if(alpha>=0 && alpha<pi/2)
        begin
        x_left=x_center*float_transfer - (length / 2) * cos_alpha - half_thick*sin_alpha;
        x_right=x_center*float_transfer + (length / 2) * cos_alpha    + half_thick*sin_alpha;
        y_left=y_center*float_transfer - (length / 2) * sin_alpha    - half_thick*cos_alpha;
        y_right=y_center*float_transfer + (length / 2) * sin_alpha    + half_thick*cos_alpha;
        end
        else if(alpha>=pi/2 && alpha<pi)
        begin
        x_left=x_center*float_transfer + (length / 2) * cos_alpha    - half_thick*sin_alpha;
        x_right=x_center*float_transfer - (length / 2) * cos_alpha + half_thick*sin_alpha;
        y_left=y_center*float_transfer - (length / 2) * sin_alpha    + half_thick*cos_alpha;
        y_right=y_center*float_transfer + (length / 2) * sin_alpha    - half_thick*cos_alpha;
        end
        else;


        x_cir1=x_center*float_transfer + (length / 2) * cos_alpha;
        y_cir1=y_center*float_transfer - (length / 2) * sin_alpha;
        x_cir2=x_center*float_transfer - (length / 2) * cos_alpha;
        y_cir2=y_center*float_transfer + (length / 2) * sin_alpha;
```

```
        //dis=(y_center - v_count) * cos_alpha + (x_center - h_count)* sin_alpha;
        //dis2=(y_center - v_count) * sin_alpha - (x_center - h_count)* cos_alpha;

        end


    //修改  x_center 和 y_cneter 的值
    //应该最终改成如果有输入信号的变化，，上下左右或者旋转

    always @ (posedge clk_100HZ)
    begin
if(is_end==1)
begin
    x_center=x_center;
    y_center=y_center;
end

else
begin
    dis_temp1=(y_ball - y_center) * cos_alpha + (x_ball - x_center)* sin_alpha;
    dis_temp2=(y_ball - y_center) * sin_alpha - (x_ball - x_center)* cos_alpha;
    /*if((dis_temp1<=(r_ball + half_thick)*float_transfer) && (dis_temp1 >= -(r_ball + half_thick)*float_transfer) && (dis_temp2<=(float_transfer*length/2)) && (dis_temp2 >= -(float_transfer*length/2)))
    begin

    x_center=x_center;
    y_center=y_center;

    end

    else
    begin*/

    if(is_move==1)
    begin
        if(move==2'b00)//红色，向左
            if(x_center<Line_base + H_ACTIVE_TIME/10 + length / 2 + 20)
            x_center=x_center;
            else
            x_center=x_center-1;
        else if(move==2'b11) //黑色，向右
            if(x_center>Line_base + 9*H_ACTIVE_TIME/10 - length / 2 - 20)
            x_center=x_center;
```

```
                else
                x_center=x_center+1;
            else ;

    end
    else;//白色，不动


    if(is_move==1)
    begin
        if(move==2'b01)//绿色  向下
            if(y_center>Column_base + 9*V_ACTIVE_TIME/10 - half_thick - 20)
            y_center=y_center;
            else
            y_center=y_center+1;
        else if(move==2'b10)//蓝色，向上
            if(y_center<Column_base + V_ACTIVE_TIME/10 + half_thick + 20)
            y_center=y_center;
            else
            y_center=y_center-1;
        else;

    end
     else;//白色，不动

    //end//与是否相撞的对应
end
    end

    //修改球的角度
    always @(posedge clk_100HZ)
    begin

if(is_end==1)
theta=theta;

else
begin
    if(x_ball<=Line_base + H_ACTIVE_TIME/10 &&cos_theta<0)
    theta=pi-theta;
    else if(x_ball>=Line_base + 9*H_ACTIVE_TIME/10 &&cos_theta>0)
    theta=pi-theta;
    else if(y_ball<=Column_base + V_ACTIVE_TIME/10 && sin_theta<0)
    theta=2*pi-theta;
```

```
    else if(y_ball>=Column_base + 9*V_ACTIVE_TIME/10 && sin_theta>0)
    theta=2*pi-theta;
    else if((dis_temp1<=(r_ball + half_thick)*float_transfer) && (dis_temp1 >= -(r_ball + half_thick)*float_transfer) && (dis_temp2<=(float_transfer*length/2)) && (dis_temp2 >= -(float_transfer*length/2)))
    theta=2*pi-theta-alpha;
    else;


    if(theta<0)
    theta=theta+2*pi;
    else if(theta>2*pi)
    theta=theta-2*pi;

end

        if((theta>=0&&theta<pi/2)||(theta>=pi&&theta<3*pi/2))
          begin
          if(theta>=0&&theta<pi/2)
            temp_theta=theta;
            else if(theta>=pi&&theta<3*pi/2)
            temp_theta=theta-pi;
            else;


sin_theta=((temp_theta)-float_transfer*temp_theta*temp_theta*temp_theta/(float_transfer*float_transfer*float_transfer)/6+float_transfer*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/120-float_transfer*(temp_theta*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/5040);

cos_theta=(float_transfer-float_transfer*(temp_theta*temp_theta)/(float_transfer*float_transfer)/2+float_transfer*(temp_theta*temp_theta*temp_theta*temp_theta)/(float_transfer*float_transfer*float_transfer*float_transfer)/24-float_transfer*(temp_theta*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/720);

          if(theta>=pi&&theta<3*pi/2)
          begin
          sin_theta=-sin_theta;
          cos_theta=-cos_theta;
          end
```

```
        end
        else if((theta>=pi/2&&theta<pi)||(theta>=3*pi/2)&&theta<2*pi)
        begin

        if(theta>=pi/2&&theta<pi)
          temp_theta=theta-pi/2;
          else if(theta>=3*pi/2&&theta<2*pi)
          temp_theta=theta-pi/2-pi;
          else;


sin_theta=(float_transfer-float_transfer*(temp_theta*temp_theta)/(float_transfer*float_transfer)/2
+float_transfer*(temp_theta*temp_theta*temp_theta*temp_theta)/(float_transfer*float_transfer*fl
oat_transfer*float_transfer)/24-float_transfer*(temp_theta*temp_theta*temp_theta*temp_theta*te
mp_theta*temp_theta)/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*f
loat_transfer)/720);

cos_theta=-((temp_theta)-float_transfer*temp_theta*temp_theta*temp_theta/(float_transfer*float_
transfer*float_transfer)/6+float_transfer*temp_theta*temp_theta*temp_theta*temp_theta*temp_th
eta/(float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/120-float_transfer*(
temp_theta*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta*temp_theta)/(float_trans
fer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer*float_transfer)/5040)
;

        if(theta>=3*pi/2&&theta<2*pi)
          begin
          sin_theta=-sin_theta;
          cos_theta=-cos_theta;
          end

          end

            else;


    end

    //修改球坐标
    reg [10:0]x_counter=1;
    reg [10:0]y_counter=1;
    reg signed [10:0]x_accu=0;
    reg signed [10:0]y_accu=0;
    reg signed [18:0]x_100ball=(Line_base + H_ACTIVE_TIME/2)*100;
    reg signed [18:0]y_100ball=(Column_base + V_ACTIVE_TIME/2 - 60)*100;
```

```
    wire clk_50HZ;

Divider #2000000 divide2(clock,rst,clk_50HZ);//分频器，分成 100Hz

    always@(posedge clk_50HZ)
    begin

if(is_end==1)
begin
x_100ball=x_100ball;
y_100ball=y_100ball;
end
else
begin
        if(x_ball<=Line_base + H_ACTIVE_TIME/10&&cos_theta<0)
        x_100ball=(Line_base + H_ACTIVE_TIME/10+1)*100;

    else if(x_ball>=Line_base + 9*H_ACTIVE_TIME/10&&cos_theta>0)
        x_100ball=(Line_base + 9*H_ACTIVE_TIME/10-1)*100;
    else if(y_ball<=Column_base + V_ACTIVE_TIME/10&&sin_theta<0)
        y_100ball=(Column_base + V_ACTIVE_TIME/10+1)*100;

    else if(y_ball>=Column_base + 9*V_ACTIVE_TIME/10&&sin_theta>0)
        y_100ball=(Column_base + 9*V_ACTIVE_TIME/10-1)*100;
    else if((dis_temp1<=(r_ball + half_thick)*float_transfer) && (dis_temp1 >= -(r_ball + half_thick)*float_transfer) && (dis_temp2<=(float_transfer*length/2)) && (dis_temp2 >= -(float_transfer*length/2)))
    begin
    x_counter=1;
    y_counter=1;
    end
    else
     begin

    x_100ball=x_100ball+cos_theta;
    y_100ball=y_100ball+sin_theta;
    end
end

    x_ball=x_100ball/100;
    y_ball=y_100ball/100;

    end
```

```
//修改 is_end 的值
always @(*)
begin

if(sig==0)
begin
if(x_ball>=Line_base + 9*H_ACTIVE_TIME/10-1 && y_ball>=Column_base + 4*V_ACTIVE_TIME/10&&y_ball<=Column_base + 6*V_ACTIVE_TIME/10)
is_end=1;
else
is_end=0;
end
else
is_end=0;


end



    //改动 RGB 的值，进行输出
    always @ (*)
    begin
    if(active_flag)
    begin
            //专门描绘边缘
            if((v_count>Column_base + V_ACTIVE_TIME/10) && (v_count<Column_base + 9*V_ACTIVE_TIME/10)&&((h_count<Line_base + H_ACTIVE_TIME/10) || (h_count>Line_base + 9*H_ACTIVE_TIME/10 && (v_count<Column_base + 4*V_ACTIVE_TIME/10 || v_count>Column_base + 6*V_ACTIVE_TIME/10))))
            begin  //边缘蓝色（留一个空）
                    if(is_move==1)
                        begin

                            if(move==2'b00)//红色，向左
                            begin
                            Red   =  4'b1111    ;
                            Green =  4'b0000    ;
                            Blue  =  4'b0000    ;

                            end
                            else if(move==2'b11) //黑色，向右
                            begin
```

```
                                    Red     =   4'b0111      ;
                                    Green =   4'b0011     ;
                                    Blue    =   4'b0001      ;

                                    end
                                    else if(move==2'b01)//绿色  向下
                                    begin
                                    Red     =   4'b0000      ;
                                    Green =   4'b1111     ;
                                    Blue    =   4'b0000      ;

                                    end
                                    else if(move==2'b10)//蓝色，向上
                                    begin
                                    Red     =   4'b0000      ;
                                    Green =   4'b0000     ;
                                    Blue    =   4'b1111      ;

                                    end
                                    else;
                                end
                                else
                                begin
                                    Red     =   4'b1111      ;
                                    Green =   4'b1111     ;
                                    Blue    =   4'b1111      ;

                                end
                        //Red     =   4'b0000      ;
                        //Green =   4'b1111     ;
                        //Blue    =   4'b0000      ;
                end
                else if((v_count<Column_base + V_ACTIVE_TIME/10) || (v_count>Column_base
+ 9*V_ACTIVE_TIME/10) )
                    begin   //边缘蓝色
                                if(is_move==1)
                                    begin

                                        if(move==2'b00)//红色，向左
                                        begin
                                        Red     =   4'b1111      ;
                                        Green =   4'b0000     ;
                                        Blue    =   4'b0000      ;
```

```
                    end
                    else if(move==2'b11) //黑色，向右
                    begin
                    Red     =   4'b0111      ;
                    Green =   4'b0011    ;
                    Blue    =   4'b0001      ;

                    end
                    else if(move==2'b01)//绿色  向下
                    begin
                    Red     =   4'b0000      ;
                    Green =   4'b1111    ;
                    Blue    =   4'b0000      ;

                    end
                    else if(move==2'b10)//蓝色，向上
                    begin
                    Red     =   4'b0000      ;
                    Green =   4'b0000    ;
                    Blue    =   4'b1111      ;

                    end
                    else;
                end
                else
                begin
                    Red     =   4'b1111      ;
                    Green =   4'b1111    ;
                    Blue    =   4'b1111      ;

                end
            //Red     =   4'b0000      ;
            //Green =   4'b1111    ;
            //Blue    =   4'b0000      ;

        end

else
begin

    //画板子（边上是圆角）

        //先画两边的球
        dis=(y_center - v_count) * cos_alpha + (x_center - h_count)* sin_alpha;
```

```
dis2=(y_center - v_count) * sin_alpha - (x_center - h_count)* cos_alpha;

if((h_count*float_transfer - x_cir1)*(h_count*float_transfer - x_cir1) + (v_count*float_transfer - y_cir1) *(v_count*float_transfer - y_cir1) <= half_thick * half_thick*float_transfer*float_transfer )
begin
    Red   =  4'b1111    ;
    Green =  4'b0000    ;
    Blue  =  4'b0000    ;
end
else if((h_count*float_transfer - x_cir2)*(h_count*float_transfer - x_cir2) + (v_count*float_transfer - y_cir2) *(v_count*float_transfer - y_cir2) <= half_thick * half_thick*float_transfer*float_transfer )
begin
    Red   =  4'b1111    ;
    Green =  4'b0000    ;
    Blue  =  4'b0000    ;
end

//else  if(h_count*float_transfer >= x_left && (h_count*float_transfer <=x_right ) && (v_count*float_transfer >= y_left ) && (v_count*float_transfer <=y_right ))
//begin
    //if((v_count - y_center) * cos_alpha + (h_count - x_center)* sin_alpha >0)//一开始窄线是因为 cos 为 500


        else  if((dis <= half_thick*float_transfer && dis>= -half_thick*float_transfer)&&( dis2 >= -float_transfer*length/2 && dis2 <= float_transfer*length/2))
        begin

            Red   =  4'b1111    ;
            Green =  4'b0000    ;
            Blue  =  4'b0000    ;
        end
//end

    //画球
else if((h_count - x_ball)*(h_count - x_ball) + (v_count - y_ball)*(v_count - y_ball)<=r_ball*r_ball/*1000000*/)
        begin

            Red   =  4'b0000    ;
           Green =  4'b1111    ;
```

```
                    Blue    =    4'b0000       ;
                end

            //其他地方
        else
                begin
                    Red     =    4'b0000       ; //  黑色
                    Green =    4'b0000     ;
                    Blue    =    4'b0000       ;
                end

    end

end

end

endmodule
```