同济大学计算机系

数字逻辑课程综合实验报告

学　　号　1652968

姓　　名　汤晨宇

专　　业　工科试验班（计算机类）

授课老师　张冬冬

目录

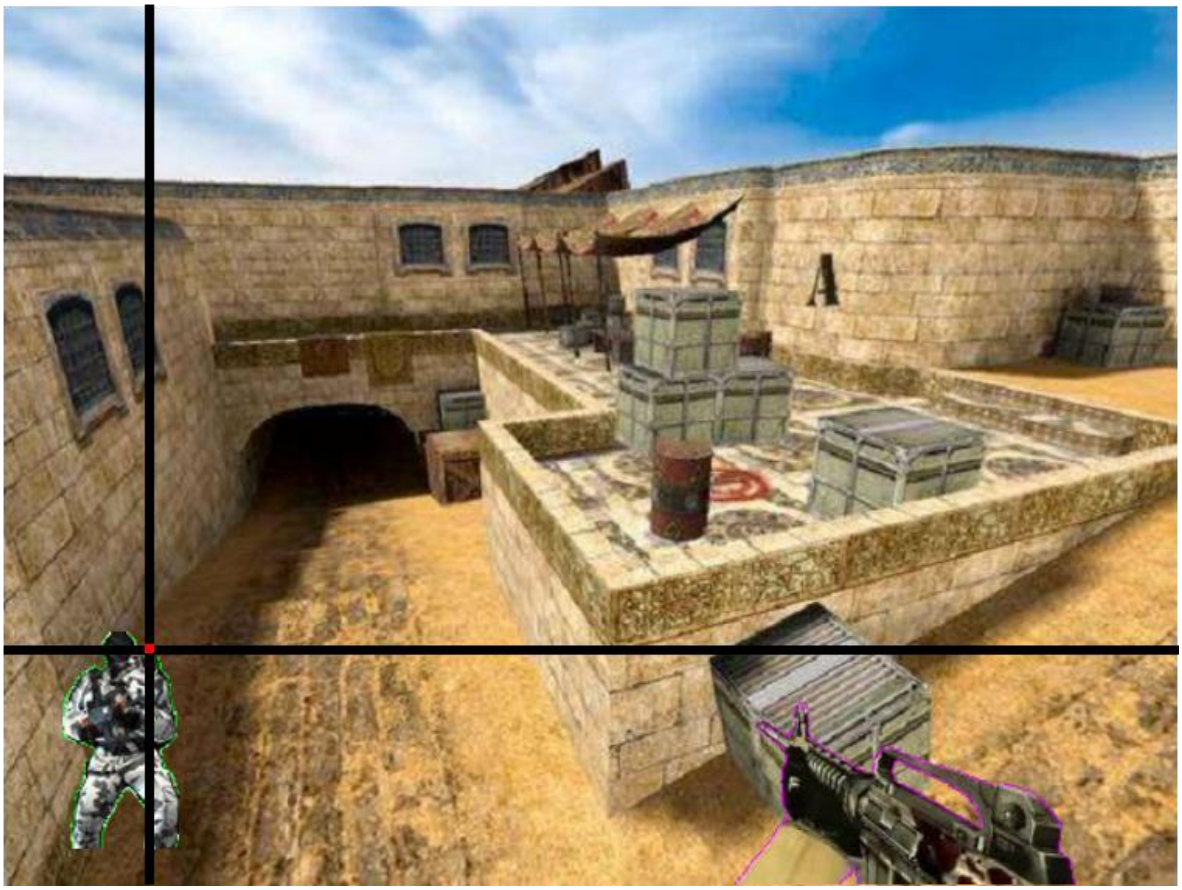
一、实验内容

1. 项目内容概括

基于 FPGA 以及 L3G4200D 三轴数字输出陀螺仪的一款模拟 CS 射击的小游戏。

2. 界面样式

如图所示，界面包括背景地图（静止），敌人（随机位置出现），准星（可随开发板转动移动）以及枪口（可随开发板转动移动）。



3. 操作说明

通过转动连接有三轴陀螺仪的 FPGA 开发板，可以移动画面中的“准星”。当准星对准敌人时，可以按下开枪键，完成对敌人的击杀。为了便于操作，另外添加了一个复位键，可以不用旋转开发板而使准星直接移回屏幕中央。

4. 规则说明

打开游戏开关后，计时开始。屏幕上开始出现敌人，可以通过转动开发板击杀敌人。每完成一次击杀，分数加一，同时在屏幕随机位置新出现一个敌人。在规定时间内（原代码为 45s），需要尽可能得到最多的分数。其中分数和倒计时会实时显示在开发板的数码管上。

5. 器件简介

L3G4200D 三轴数字输出陀螺仪——由意法半导体（ST）集团制造的一款三轴数字输出陀螺仪。用于测量空间三个维度的角速度，同时支持 SPI 和 I^2C 协议。在本次综合实验中，选择的是 SPI 协议

VGA——VGA(Video Graphics Array)是 IBM 在 1987 年随 PS/2 机一起推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点；

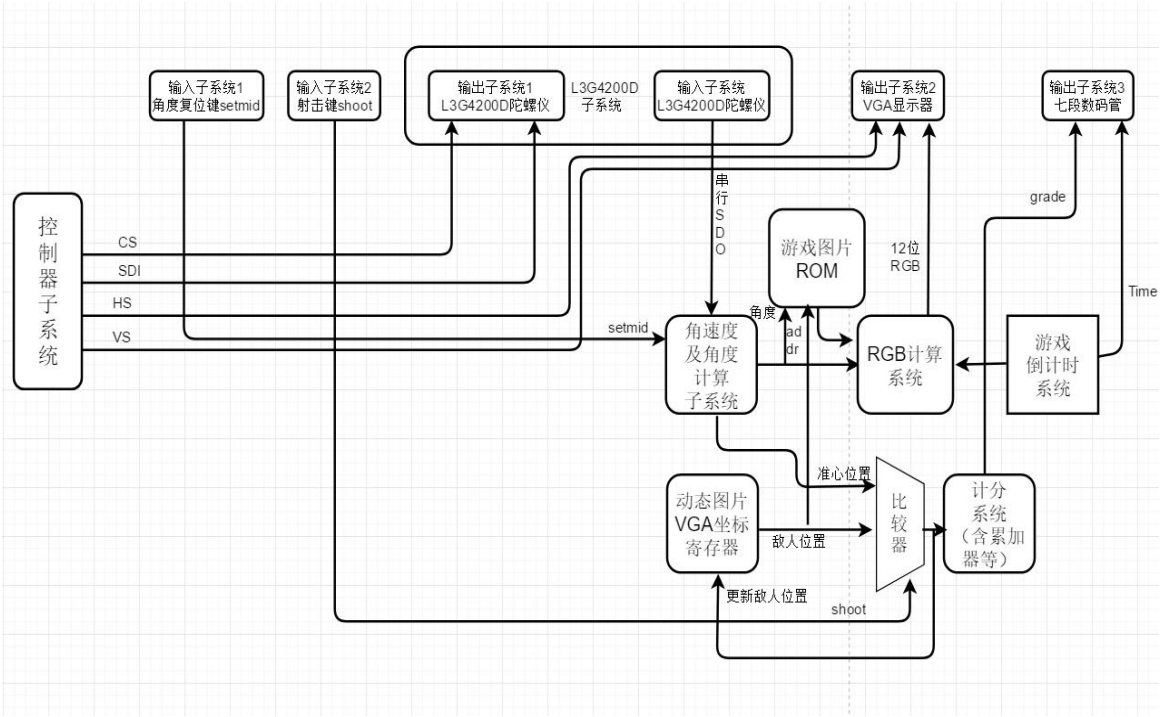
Nexys 4 DDR Artix-7——由 Xilinx 公司开发出的一款现场可编程门阵列（FPGA）开发板。

二、射击小游戏数字系统总框图

初步分析，需要完成的功能是大致有如下几点：

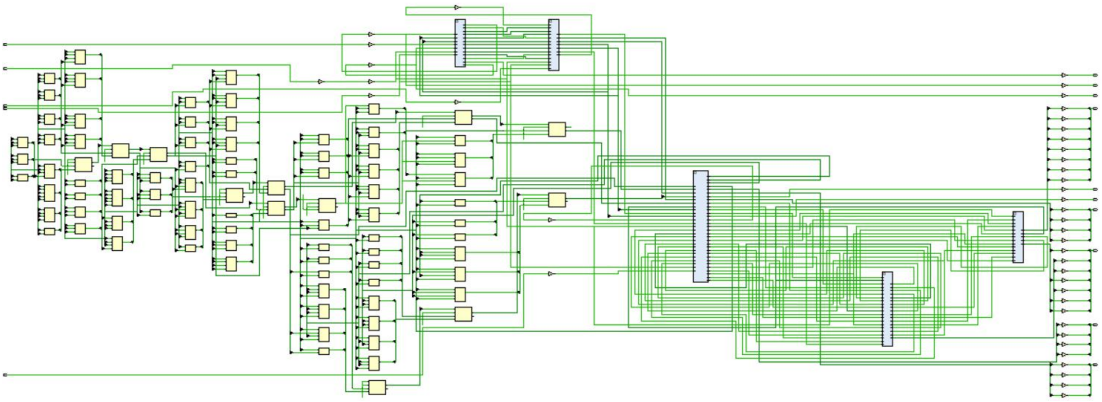
- 向 L3G4200D 输出写寄存器与读寄存器指令；
- 从 L3G4200D 输入三轴陀螺仪读数（角速度）；
- 处理读取到的角速度，使其能反应转动的角度；
- 有角度复位按钮，可以在按下的时候角度设置为 0° 的初始情况；
- 将角度转换为准心以及枪口的位置，在 VGA 上显示；
- 设置存储各类图片的 ROM；
- 向 VGA 输出正确的 HS 与 VS 时序，使 VGA 能正确显示图片；
- 对游戏倒计时，在七段数码管上显示；
- 时间到了之后，屏幕上显示 Game Over 的图片提示，倒计时停止在 0 不再变动，七段数码管上除了最终分数其余位均为 0；
- 有一个随机生成并存储敌人最新位置的机制；
- 按下开枪键后，判断准心是否与敌人重合，若重合则加分并重新生成敌人位置，否则不会触发任何时间。分数也会在七段数码管上实时显示；
- 根据之前准心的位置、敌人的位置、枪口的位置，决定访问哪一个图片的 ROM，以及应该读取的地址，按照相应时序调整 12 位 RGB 数据；
- ……

综上所述，设计了如下所示的总框图——



如图所示，可以大致反映系统的逻辑功能。

RTL 图片如下图所示：



详细模块参考下方截图。

三、系统控制器设计

将由于 VGA 显示器模块和 L3G4200D 数据读取和处理频率相差较大（例如显示器画面更新频率仅为 60Hz，但是数据读取可能有几万赫兹，不是同步进行的），相互影响较少，且状态较多，因此此处将若干个相互影响较少的子系统分别独立列出。

将按照以下顺序列出状态转移表以及 ASM 流程图。

①L3G4200 读写

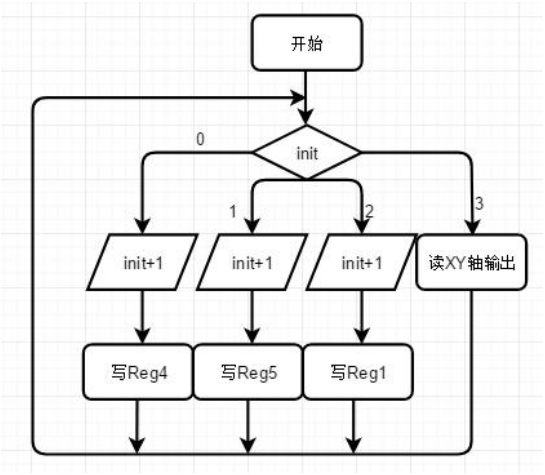
开始(100)	写 Reg4(000)	init = 00
写 Reg4(000)	写 Reg5(001)	init = 01
写 Reg5(001)	写 Reg1(010)	init = 10
写 Reg1(010)	读 XY 轴输出(011)	init = 11
读 XY 轴输出(011)	读 XY 轴输出(011)	init = 11

可知~rst 即为状态码第 1 位，init 位状态码后两位。

$C^{n+1} = 0$

$B^{n+1} = B^n + A^n$

$A^{n+1} = \overline{C}^n(\overline{B}^n + A^n)$



②SPI 协议相关部分

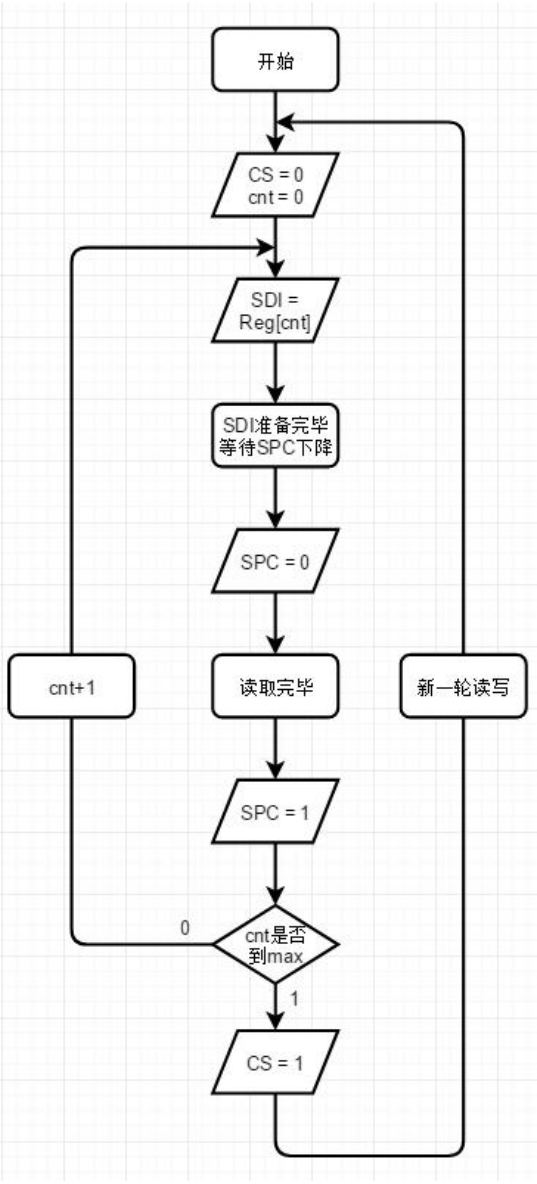
PS	NS	转换条件
开始(00)	SDI 准备(01)	/
SDI 准备(01)	读取完毕(10)	/
读取完毕(10)	SDI 准备(01)	cnt 未到 max
读取完毕(10)	新一轮读写(11)	cnt 到 max
新一轮读写(11)	SDI 准备(01)	/

计 X 表示 cnt 是否到最大值，到为 1，未到为 0，则

$B^{n+1} = \overline{B}^n A^n + XB^n\overline{A}^n$

$A^{n+1} = \overline{B}^n \oplus A^n + \overline{X}B^n\overline{A}^n$

流程图如下：



③角度计算测量部分

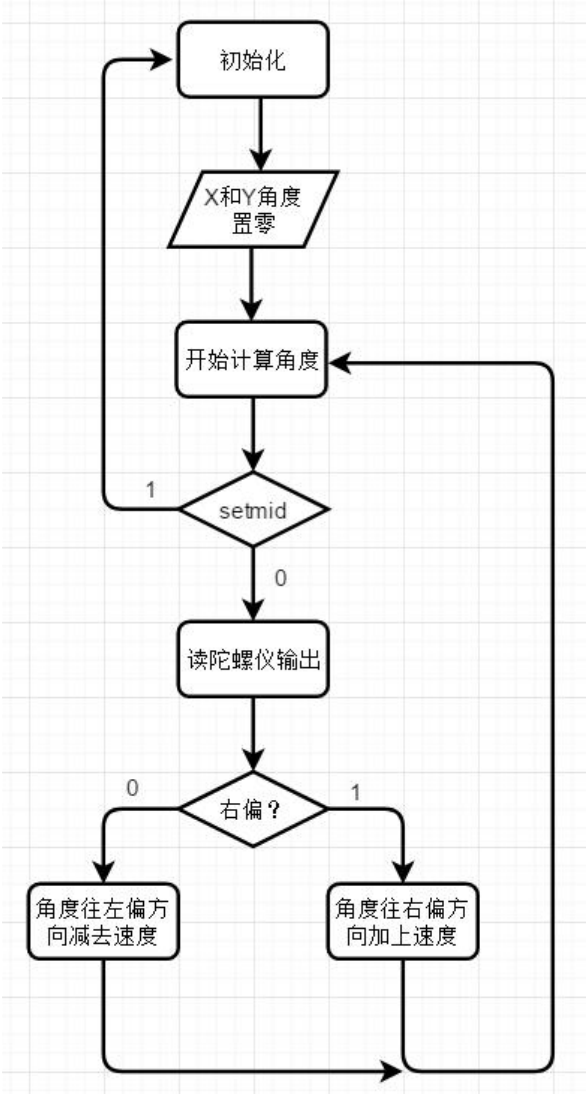
PS	NS	转换条件
初始化(000)	开始计算角度(001)	/
开始计算角度(001)	初始化(000)	setmid = 1
开始计算角度(001)	读陀螺仪输出(010)	setmid = 0
读陀螺仪输出(010)	角度往左减(011)	读数不右偏
读陀螺仪输出(010)	角度往右加(100)	读数右偏
角度往左减(011)	开始计算角度(001)	/
角度往右加(100)	开始计算角度(001)	/

计 setmid 的值为 X，读数是否右偏为 Y（右偏为 1）

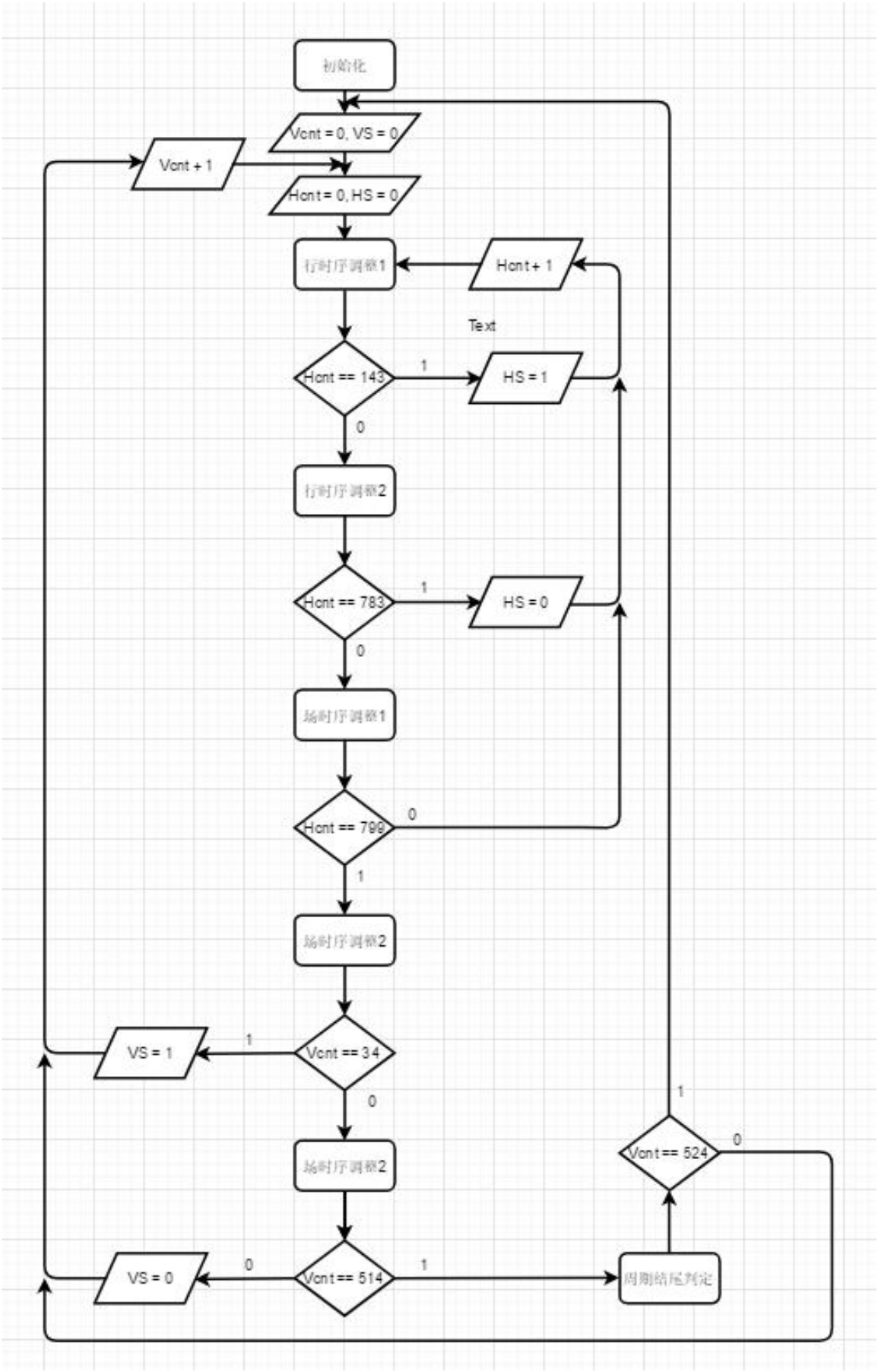
$$C^{n+1} = Y\overline{C}^{n}B^{n}\overline{A}^{n}$$

$$B^{n+1} = \overline{C}^{n}(\overline{XB}^{n}A^{n} + \overline{Y}B^{n}\overline{A}^{n})$$

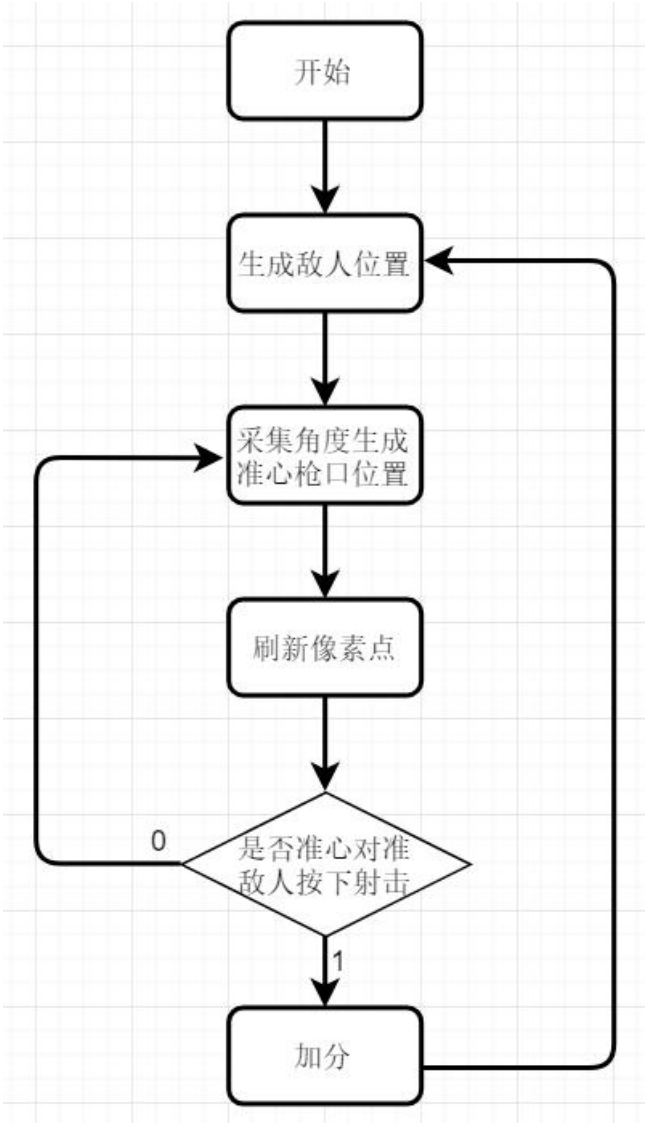
$$A^{n+1} = \overline{B}^{n}\overline{A}^{n} + \overline{C}^{n}B^{n}(A^{n} + Y\overline{A}^{n})$$



④HS 和 VS 信号设计


初始化
Vcnt = 0, VS = 0
Vcnt + 1
Hcnt = 0, HS = 0
行时序调整1
Hcnt + 1
Text
Hcnt == 143
1
HS = 1
0
行时序调整2
Hcnt == 783
1
HS = 0
0
场时序调整1
Hcnt == 799
0
1
场时序调整2
VS = 1
1
Vcnt == 34
0
场时序调整2
1
Vcnt == 524
0
VS = 0
0
Vcnt == 514
1
周期结尾判定

⑤画面、计分及游戏逻辑



PS	NS	转换条件
开始(100)	生成敌人位置(000)	/
生成敌人位置(000)	采集角度信息(001)	/
采集角度信息(001)	刷新像素点(010)	/
刷新像素点(010)	采集角度信息(001)	对准敌人射击
刷新像素点(010)	加分(011)	无有效射击
加分(011)	生成敌人位置(000)	/

计是否对准完成一次射击的值为 X，射击成功为 1，否则为 0.
实质上 C 为即为~rst。

$$C^{n+1} = 0$$

$$B^{n+1} = \overline{B}^{n}\overline{A}^{n} + XB^{n}\overline{A}^{n} + B^{n}A^{n}$$

$$A^{n+1} = \overline{C}^{n}\overline{A}^{n}$$

四、子系统模块建模

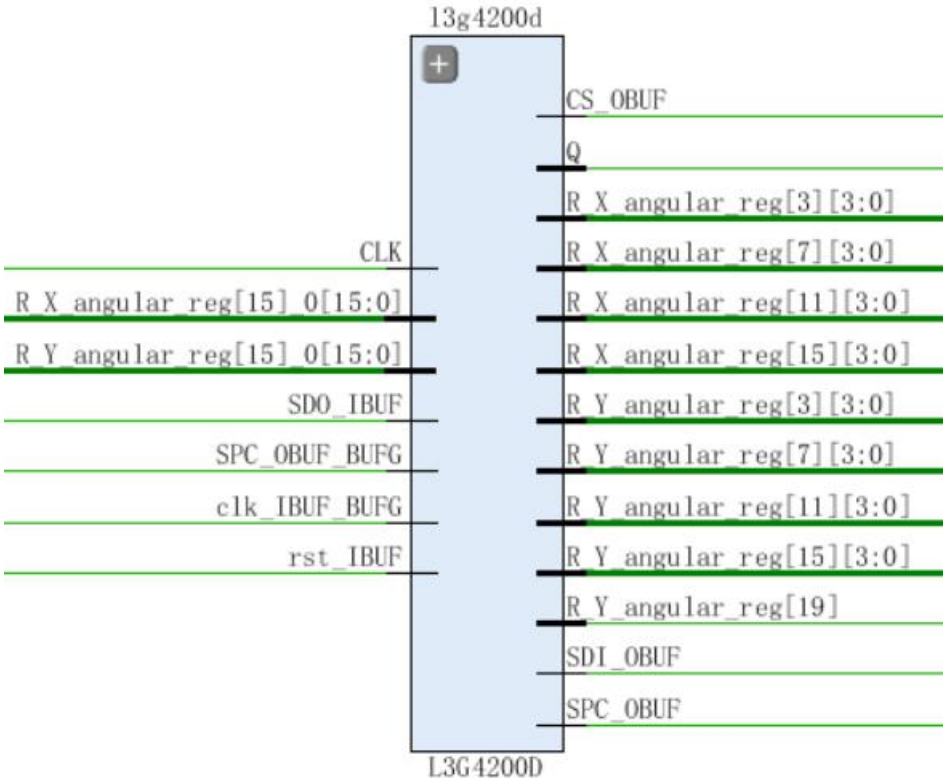
1. *Game 顶层模块*

顶层模块，连接各个子模块，列出了所有的与外部交流的数据。

```
module Game(
    input clk,          //系统时钟
    input rst,          //游戏开关
    input setmid,       //角度复位键
    input shoot,        //射击键
    input SDO,          //L3G4200D 读取信号
    output CS,          //L3G4200D 使能
    output SPC,         //L3G4200D 时钟
    output SDI,         //L3G4200D 传入信号
    output [6:0]oData,  //七段数码管显示数据
    output [7:0]AN,     //七段数码管选择
    output [3:0]O_RED,  //VGA 红色分量
    output [3:0]O_GREEN, //VGA 绿色分量
    output [3:0]O_BLUE,  //VGA 蓝色分量
    output O_HS,        //行时序
    output O_VS         //场时序
    );

    wire [15:0]X_data;
    wire [15:0]Y_data;
    wire [9:0]X_angular;
    wire [9:0]Y_angular;
    wire [15:0]grade;
    wire [15:0]Time;
    wire live;
    //assign light = data;
    L3G4200D l3g4200d(clk, rst, SDO, CS, SPC, SDI, X_data, Y_data);
    timecnt(clk, rst, live, Time);
    SegmentDisplay display(clk, rst, grade, Time, oData, AN);
    BlockMove bm(clk, rst, setmid, X_data, Y_data, X_angular, Y_angular);
    VGA(clk, rst, shoot, live, X_angular, Y_angular, grade, O_RED,
O_GREEN, O_BLUE, O_HS, O_VS);
endmodule
```

2. *L3G4200D 子模块*

陀螺仪数据读取模块，负责写入、初始化陀螺仪，同时读取陀螺仪传出的角速度。




```
module L3G4200D(
    input CLK100MHz,
    input rst,
    input sdo,
    output cs,
    output spc,
    output sdi,
    output reg [15:0]X_data,
    output reg [15:0]Y_data
    );

    parameter WriteReg1 = 16'b00100000_11111111;
    parameter WriteReg4 = 16'b00100011_00010000;
    parameter WriteReg5 = 16'b00100100_00010000;
    parameter Read_OUT_X = 8'b11101000;
    parameter CLK100MHz_MAXCOUNT = 600;
    parameter CS_MAXCOUNT = CLK100MHz_MAXCOUNT * 164;  //CS 分频计数
    reg SDI = 1'b1;
    reg CS = 1'b1;
    reg SPC = 1'b1;
    reg [15:0] X_data_reg = 0;
    reg [15:0] Y_data_reg = 0;
    reg [15:0] X_delta = 0;
    reg [15:0] Y_delta = 0;
```

```
    reg [23:0] X_deltasum = 24'b10000000_00000000_00000000;
    reg [23:0] Y_deltasum = 24'b10000000_00000000_00000000;
    reg [7:0] count_delta = 0;
    reg [23:0] count_CS = 0;  //CS 分频
    reg [23:0] count_CLK100MHz = 0;
    reg [7:0] count_SPC = 81;   //82 等分，每 2 个一个周期
    reg [1:0]init = 0;   //是否初始化
    assign sdi = SDI;
    assign cs = CS;
    assign spc = SPC;

    /* CS 信号生成 */
    always@(posedge CLK100MHz or negedge rst) begin
        if(rst == 1'b0) begin
        /* 开关关闭，CS 置为高电平，CS 计数器置 0 */
            CS <= 1;
            count_CS <= 0;
        end
        else begin
        /* 开关打开 */
            if(count_CS >= CS_MAXCOUNT - 1) begin
            /* CS 计数器达到目标值，CS 反相，计数器置零 */
                count_CS <= 0;
                CS <= ~CS;
            end
            else begin
            /* CS 计数器未到目标值，继续累加 */
                count_CS <= count_CS + 1;
            end
        end
    end

    /* 判断是否完成初始化 */
    always@(posedge CS or negedge rst) begin
        if(rst == 1'b0)
            init <= 0;
        else
            if (init < 3)
                init <= init + 1;
            else
                init <= 3;
    end

    /*********************************
```

```
    SPC 分频计数，count_SPC 初始为 49，81
    80,46,...,2 为低电平(最多 40 个)
    81,47,...,1 为高电平(最多 41 个)
    ********************************/
    always@(posedge CLK100MHz or negedge rst or negedge CS) begin
        if(rst == 1'b0 || CS == 1'b1) begin
        /* 开关关闭或 CS 高电平，CLK100MHz 计数器置 0 */
            count_CLK100MHz <= 0;
            count_SPC <= 81;
        end
        else begin
        /* 开关打开后 CS 低电平状态下 CLK100MHz 上升沿触发 */
            if(count_CLK100MHz >= CLK100MHz_MAXCOUNT) begin
            /* CLK100MHz 计数器到达目标值，SPC 计数-1，CLK100MHz 计数器归 0 */
                count_SPC <= count_SPC - 1;
                count_CLK100MHz <= 0;
            end
            else
            /* CLK100MHz 计数器还未到达目标值，继续累加 */
                count_CLK100MHz <= count_CLK100MHz + 1;
        end
    end

    //生成 SPC 脉冲
    always@(count_SPC or rst or init) begin
        if(rst == 1'b0 || count_SPC == 81)
        /* 开关关闭或 SPC 计数器在初始状态，SPC 处于高电平 */
            SPC <= 1;
        else if(init < 3) begin
        /* 还未初始化 */
            if(count_SPC >= 81 || count_SPC < 49)
            /* 由于写入周期较少，之后有段时间一直高电平 */
                SPC <= 1;
            else if(count_SPC % 2 == 1)
            /* 余 2 为 1 置为高电平 */
                SPC <= 1;
            else
            /* 余 2 为 0 置为低电平 */
                SPC <= 0;
        end
        else begin
            if(count_SPC >= 81 || count_SPC < 1)
            /* 首尾均为高电平 */
                SPC <= 1;
```

```
        else if(count_SPC % 2 == 1)
        /* 余 2 为 1 置为高电平 */
            SPC <= 1;
        else
        /* 余 2 为 0 置为低电平 */
            SPC <= 0;
    end
end

//SDI 输入指令
always@(count_SPC or init) begin
    if (init == 0) begin
        if (count_SPC < 81 && count_SPC >= 49)
        /* 写入指令 */
            SDI <= WriteReg4[(count_SPC + 1) / 2 - 25];
        else
        /* 16 位以上无指令 */
            SDI <= 0;
    end
    else if(init == 1) begin
    /* 写 Reg_1 */
        if (count_SPC < 81 && count_SPC >= 49)
        /* 写入指令 */
            SDI <= WriteReg5[(count_SPC + 1) / 2 - 25];
        else
        /* 16 位以上无指令 */
            SDI <= 0;
    end
    else if(init == 2) begin
    /* 写 Reg_1 */
        if (count_SPC < 81 && count_SPC >= 49)
        /* 写入指令 */
            SDI <= WriteReg1[(count_SPC + 1) / 2 - 25];
        else
        /* 16 位以上无指令 */
            SDI <= 0;
    end
    else begin  //写入读 x 轴角速度指令
    /* 初始化完成 */
        if (count_SPC < 81 && count_SPC >= 65)
        /* 读取指令 */
            SDI <= Read_OUT_X[(count_SPC + 1) / 2 - 33];
        else
        /* 8 位以上均为 SDO 读取，无指令 */
```

```
            SDI <= 0;
        end
    end

    //读取数据
    always@(posedge SPC or negedge rst) begin
        if(rst == 1'b0) begin
        /* 开关关闭 */
            X_data_reg <= 0;
            Y_data_reg <= 0;
        end
        else begin
        /* 开关打开 */
            if(init < 3) begin
            /* 未初始化，data 为 0 */
                X_data_reg <= 0;
                Y_data_reg <= 0;
            end
            else begin
            /* 初始化完成 */
                if(count_SPC < 65 && count_SPC >= 49)
                /* 每次 SPC 下降沿读入后的上升沿写入 reg，先低位 s */
                    X_data_reg[(count_SPC + 1) / 2 - 25] <= sdo;
                else if (count_SPC < 49 && count_SPC >= 33)
                /* 后高位 */
                    X_data_reg[(count_SPC + 1) / 2 - 9] <= sdo;
                else if(count_SPC < 33 && count_SPC >= 17)
                /* 每次 SPC 下降沿读入后的上升沿写入 reg，先低位 s */
                    Y_data_reg[(count_SPC + 1) / 2 - 9] <= sdo;
                else if (count_SPC < 17 && count_SPC >= 1)
                /* 后高位 */
                    Y_data_reg[(count_SPC + 1) / 2 + 7] <= sdo;
                else begin
                /* 其余情况 reg 内数据不变 */
                    X_data_reg <= X_data_reg;
                    Y_data_reg <= Y_data_reg;
                end
            end
        end
    end

    always@(posedge CS or negedge rst) begin
    /* CS 上升沿统计误差或传值给 data */
        if(rst == 1'b0) begin
```

```
        /* 开关关闭，reg 内全 0 */
            count_delta = 0;
            X_delta = 0;
            Y_delta = 0;
            X_deltasum = 24'b10000000_00000000_00000000;
            Y_deltasum = 24'b10000000_00000000_00000000;
            X_data <= 0;
            Y_data <= 0;
        end
        else begin
        /* 统计误差 */
            if (count_delta < 126) begin
                count_delta = count_delta + 1;
                X_delta = 0;
                Y_delta = 0;
                X_deltasum = 24'b10000000_00000000_00000000;
                Y_deltasum = 24'b10000000_00000000_00000000;
                X_data <= 0;
                Y_data <= 0;
            end
            else if (count_delta < 254) begin
                count_delta = count_delta + 1;
                if (X_data_reg[15] == 1'b1) begin
                /* 向右 */
                    X_delta = 0;
                    X_deltasum = X_deltasum + 65536 - X_data_reg;
                    X_data <= 0;
                end
                else begin
                /* 向左 */
                    X_delta = 0;
                    X_deltasum = X_deltasum - X_data_reg;
                    X_data <= 0;
                end
                if (Y_data_reg[15] == 1'b1) begin
                /* 向前 */
                    Y_delta = 0;
                    Y_deltasum = Y_deltasum + 65536 - Y_data_reg;
                    Y_data <= 0;
                end
                else begin
                /* 向后 */
                    Y_delta = 0;
                    Y_deltasum = Y_deltasum - Y_data_reg;
```

```
                Y_data <= 0;
            end
        end
        else if (count_delta == 254) begin
        /* 一次性将 reg 内高低位传出 */
            count_delta = 255;
            if (X_deltasum[23] == 1'b1) begin
            /* 整体右偏 */
                X_delta = X_deltasum[22:0] / 128 + 32768;
                X_deltasum = 0;
                X_data <= 0;
            end
            else begin
            /* 整体左偏 */
                X_delta = (8388608 - X_deltasum[22:0]) / 128;
                X_deltasum = 0;
                X_data <= 0;
            end
            if (Y_deltasum[23] == 1'b1) begin
            /* 整体前偏 */
                Y_delta = Y_deltasum[22:0] / 128 + 32768;
                Y_deltasum = 0;
                Y_data <= 0;
            end
            else begin
            /* 整体后偏 */
                Y_delta = (8388608 - Y_deltasum[22:0]) / 128;
                Y_deltasum = 0;
                Y_data <= 0;
            end
        end
        else begin
            count_delta = 255;
            if (X_delta[15] == 1'b1) begin
            /* 整体右偏 */
                X_delta = X_delta;
                X_deltasum = 0;
                X_data <= X_data_reg + X_delta[14:0];
            end
            else begin
            /* 整体左偏 */
                X_delta = X_delta;
                X_deltasum = 0;
                X_data <= X_data_reg - X_delta[14:0];
```

```
                end
                if (Y_delta[15] == 1'b1) begin
                /* 整体前偏 */
                    Y_delta = Y_delta;
                    Y_deltasum = 0;
                    Y_data <= Y_data_reg + Y_delta[14:0];
                end
                else begin
                /* 整体后偏 */
                    Y_delta = Y_delta;
                    Y_deltasum = 0;
                    Y_data <= Y_data_reg - Y_delta[14:0];
                end
            end
        end
    end
endmodule
```

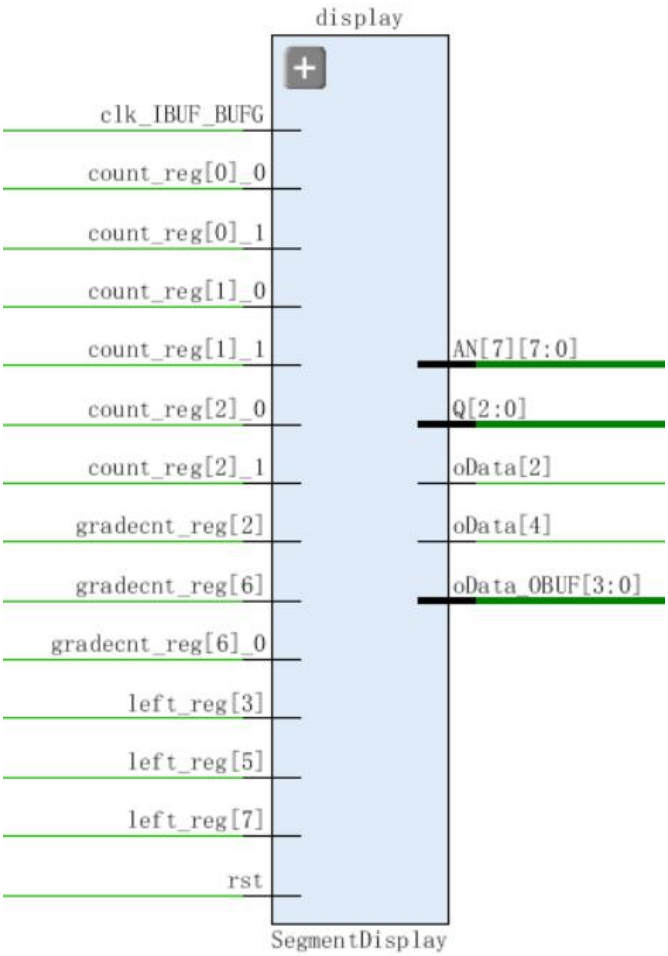
3. *timecnt 子模块*

倒计时模块，对系统时钟分频，传出剩余时间和是否游戏结束

```
module timecnt(
    input clk,
    input rst,
    output reg live,
    output reg [15:0] Time
    );
    integer counter = 0;
    reg out = 0;
    always@(posedge clk or negedge rst) begin
        if (rst == 1'b0) begin
            counter = 0;
            out <= 0;
        end
        else begin
            if (counter >= N >> 50000000) begin
                counter = 0;
                out <= ~out;
            end
            else begin
                counter = counter + 1;
                out <= out;
            end
        end
```

```
    end
    always@(posedge out or negedge rst) begin
        if (rst == 1'b0) begin
            Time <= 45;
            live <= 1;
        end
        else begin
            if (Time > 0) begin
                Time <= Time - 1;
                live <= 1;
            end
            else begin
                Time <= 0;
                live <= 0;
            end
        end
    end
endmodule
```

4. *SegmentDisplay 子模块*



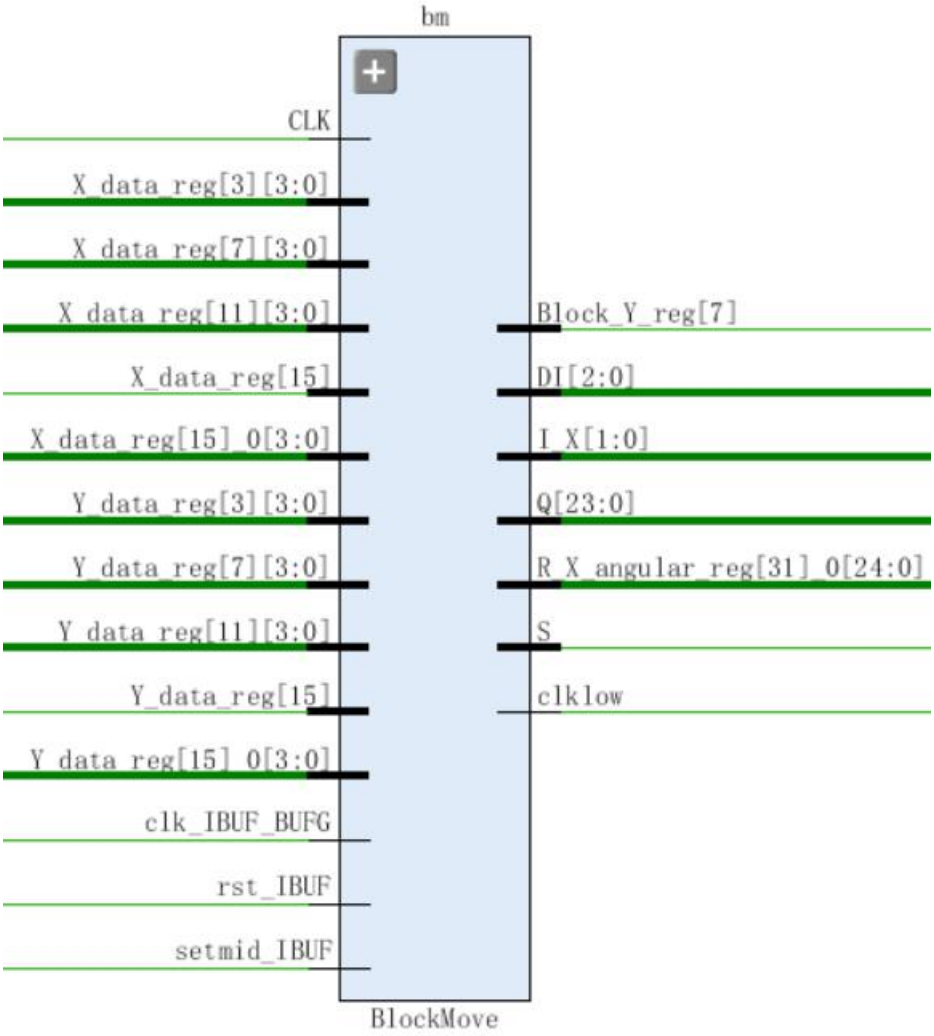
七段数码管显示模块

```
module SegmentDisplay(
    input CLK_100MHz,
    input rst,
    input [15:0]Score,
    input [15:0]Time,
    output reg [6:0]oData,
    output reg [7:0]AN
    );

    reg [2:0]count;
    reg [15:0]count_clk;
    reg CLK_1000Hz = 1'b0;
    reg [15:0]Data;  //存放 Score 和 Time
    reg [3:0]data[0:7];
    //产生 1000Hz 时钟
    always@(posedge CLK_100MHz, negedge rst) begin
        if(!rst) begin
            count_clk <= 0;
            CLK_1000Hz <= 0;
        end
        else begin
            if(count_clk == 16'd49999) begin
                count_clk <= 16'b0;
                CLK_1000Hz <= ~CLK_1000Hz;
            end
            else
                count_clk <= count_clk + 1;
        end
    end

    always@(*) begin
        Data = Score;
        data[0] = Data % 10;
        Data = Data / 10;
        data[1] = Data % 10;
        Data = Data / 10;
        data[2] = Data % 10;
        Data = Data / 10;
        data[3] = Data % 10;

        Data = Time;
        data[4] = Data % 10;
        Data = Data / 10;
```

```
        data[5] = Data % 10;
        Data = Data / 10;
        data[6] = Data % 10;
        Data = Data / 10;
        data[7] = Data % 10;
        Data = Data / 10;
    end

    //位控
    always@ (posedge CLK_1000Hz, negedge rst) begin
        if(rst == 1'b0) begin
            AN = 8'b11111111;
            count = 0;
        end
        else begin
            if(count == 3'd7)
                count = 0;
            else
                count = count + 1;
            case(count)
                3'd0: AN = 8'b11111110;
                3'd1: AN = 8'b11111101;
                3'd2: AN = 8'b11111011;
                3'd3: AN = 8'b11110111;
                3'd4: AN = 8'b11101111;
                3'd5: AN = 8'b11011111;
                3'd6: AN = 8'b10111111;
                3'd7: AN = 8'b01111111;
                default: AN = 8'b11111111;
            endcase
        end
    end

    //译码
    always@(*) begin
        case(data[count])
            0: oData = 7'b1000000;
            1: oData = 7'b1111001;
            2: oData = 7'b0100100;
            3: oData = 7'b0110000;
            4: oData = 7'b0011001;
            5: oData = 7'b0010010;
            6: oData = 7'b0000010;
            7: oData = 7'b1111000;
```

```
            8: oData = 7'b0000000;
            9: oData = 7'b0010000;
            default oData=7'b0;
        endcase
    end
endmodule
```

5. *BlockMove 子模块*

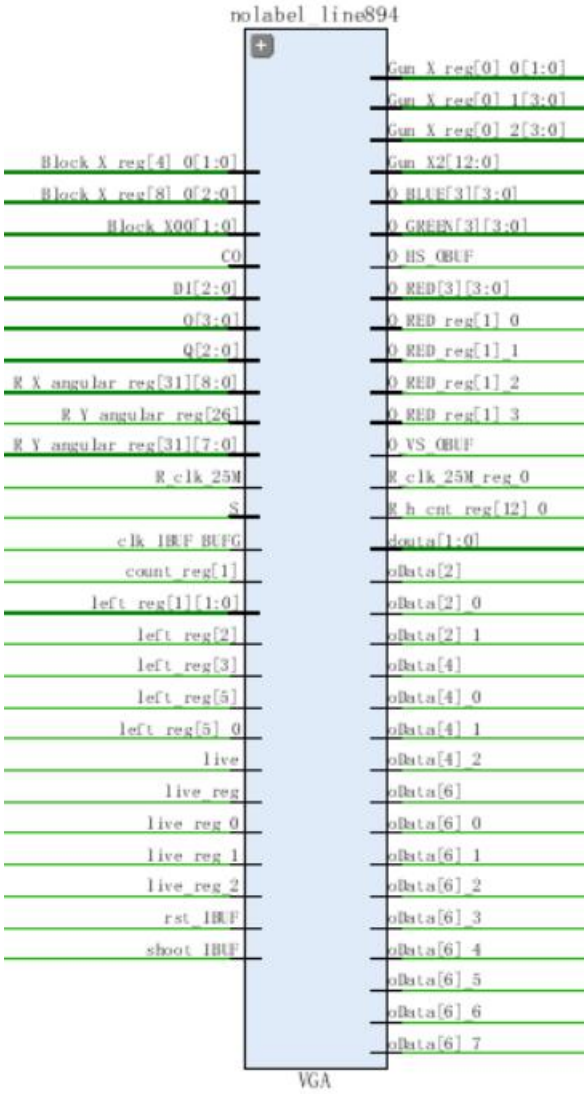



```
module BlockMove(
    input clk,
    input rst,
    input setmid,
    input [15:0]X_data,
    input [15:0]Y_data,
    output [9:0]X_angular,
    output [9:0]Y_angular
    );
```

```
reg [6:0] clkcnt = 0;
reg clklow = 1;
reg [31:0]R_X_angular = 32'b10000000_00000000_00000000_00000000;
reg [31:0]R_Y_angular = 32'b10000000_00000000_00000000_00000000;
//根据 speed 区间改变 position

always@(posedge clk) begin
    if (clkcnt < 127)
        clkcnt = clkcnt + 1;
    else begin
        clkcnt = 0;
        clklow = ~clklow;
    end
end

always@(posedge clklow or negedge rst or posedge setmid) begin
    if (rst == 1'b0) begin
        R_X_angular = 32'b10000000_00000000_00000000_00000000;
        R_Y_angular = 32'b10000000_00000000_00000000_00000000;
    end
    else begin
        if (setmid == 1'b1) begin
            R_X_angular = 32'b10000000_00000000_00000000_00000000;
            R_Y_angular = 32'b10000000_00000000_00000000_00000000;
        end
        else begin
            if (X_data[15] == 1'b1) begin
            /* 右偏 */
                R_X_angular = R_X_angular - X_data + 65536;
            end
            else begin
            /* 左偏 */
                R_X_angular = R_X_angular - X_data;
            end
            if (Y_data[15] == 1'b1) begin
            /* 右偏 */
                R_Y_angular = R_Y_angular - Y_data + 65536;
            end
            else begin
            /* 左偏 */
                R_Y_angular = R_Y_angular - Y_data;
            end
        end
```

```
        end
    end

    assign X_angular = R_X_angular[31:22];
    assign Y_angular = R_Y_angular[31:22];
endmodule
```

6. *VGA 子模块*




```
module VGA(
    input CLK_100MHz, // 系统 100MHz 时钟
    input rst, // 系统复位
    input shoot,
    input live,
    input [9:0]I_X,
```

```
    input [9:0]I_Y,
    output [15:0] grade,
    output reg [3:0]O_RED, // 红
    output reg [3:0]O_GREEN, // 绿
    output reg [3:0]O_BLUE, // 蓝
    output O_HS, // VGA 行同步信号
    output O_VS     // VGA 场同步信号
    );

    parameter C_HS_SYNC_PULSE = 96;
    parameter C_HS_BACK_PORCH = 48;
    parameter C_HS_ACTIVE_TIME = 640;
    parameter C_HS_FRONT_PORCH = 16;  //未用到，凑整强迫症列出
    parameter C_HS_LINE_PERIOD = 800;
    parameter C_VS_SYNC_PULSE = 2;
    parameter C_VS_BACK_PORCH = 33;
    parameter C_VS_ACTIVE_TIME = 480;
    parameter C_VS_FRONT_PORCH = 10;  //未用到，凑整强迫症列出
    parameter C_VS_FRAME_PERIOD = 525;

    parameter C_COLOR_BAR_WIDTH  = C_HS_ACTIVE_TIME / 8;
    parameter C_COLOR_BAR_HEIGHT = C_VS_ACTIVE_TIME / 8;

    parameter BLOCK_WIDTH = 5;
    parameter BLOCK_HEIGHT = 5;
    parameter TARGET_WIDTH = 60;
    parameter TARGET_HEIGHT = 112;

    parameter MAP_MAXCOUNT = 307200;

    reg [12:0] R_HS_count = 0; // 行时序计数器
    reg [12:0] R_VS_count = 0; // 列时序计数器
    reg [17:0] targetcount = 0;
    reg R_clk_25MHz = 1'b0;
    reg [9:0]Block_X = 0;
    reg [9:0]Block_Y = 0;
    reg [9:0]Gun_X = 0;
    reg [9:0]Gun_Y = 0;
    reg [8:0]Target_X = 0;
    reg [8:0]Target_Y = 0;
    reg [15:0]gradecnt = 0;
    assign grade = gradecnt;

    reg stopshoot = 0;
```

```
    wire [11:0]MAN_COLOR;
    wire [11:0]MAP_COLOR;
    wire [11:0]OVER_COLOR;
    wire [11:0]GUN_COLOR;
    wire [12:0]MAN_ADDR;
    wire [15:0]OVER_ADDR;
    wire [14:0]GUN_ADDR;
    reg [18:0]MAP_ADDR = 0;
    assign MAN_ADDR = (R_HS_count - C_HS_SYNC_PULSE - C_HS_BACK_PORCH -
Target_X) + (R_VS_count - C_VS_SYNC_PULSE - C_VS_BACK_PORCH - Target_Y)
* TARGET_WIDTH;
    assign OVER_ADDR = (R_HS_count - C_HS_SYNC_PULSE - C_HS_BACK_PORCH)
+ (R_VS_count - C_VS_SYNC_PULSE - C_VS_BACK_PORCH - 200) *
C_HS_ACTIVE_TIME;
    assign GUN_ADDR = (R_HS_count - C_HS_SYNC_PULSE - C_HS_BACK_PORCH -
Gun_X) + (R_VS_count - C_VS_SYNC_PULSE - C_VS_BACK_PORCH - Gun_Y) * 192;
    wire Active_Flag; // 激活标志，当这个信号为 1 时 RGB 的数据可以显示在屏幕上


    MAN_ROM man(CLK_100MHz, MAN_ADDR, MAN_COLOR);
    MAP_ROM map(CLK_100MHz, MAP_ADDR, MAP_COLOR);
    OVER_ROM over(CLK_100MHz, OVER_ADDR, OVER_COLOR);
    GUN_ROM gun(CLK_100MHz, GUN_ADDR, GUN_COLOR);


    // 产生 25MHz 的像素时钟
    reg count = 1'b0;
    always @(posedge CLK_100MHz) begin
        if(count == 1'b1) begin
            count = 1'b0;
            R_clk_25MHz <=  ~R_clk_25MHz;
            targetcount = targetcount + 1;
        end
        else
            count <= 1'b1;
    end

    always @(posedge R_clk_25MHz or negedge rst) begin
        if(!rst) begin
            Block_X <= 0;
            Block_Y <= 0;
            Gun_X <= 380;
            Gun_Y <= 370;
        end
```

```
        else begin
            if (R_HS_count < C_HS_SYNC_PULSE && R_VS_count <
C_VS_SYNC_PULSE) begin
                Block_X <= I_X / 2 + I_X / 8;
                Block_Y <= I_Y / 4 + I_Y / 8 + I_Y / 16 + I_Y / 32;
                Gun_X <= 380 + Block_X * 7 / 20;
                Gun_Y <= 370 + Block_Y / 5;
            end
            else begin
                Block_X <= Block_X;
                Block_Y <= Block_Y;
                Gun_X <= Gun_X;
                Gun_Y <= Gun_Y;
            end
        end
    end

    always @(posedge R_clk_25MHz or negedge rst or posedge shoot) begin
        if(!rst) begin
            Target_X <= targetcount[17:9] + 50;
            Target_Y <= targetcount[8:0] / 2 + 50;
            gradecnt = 0;
            stopshoot = 0;
        end
        else begin
            if(!shoot) begin
                Target_X <= Target_X;
                Target_Y <= Target_Y;
                gradecnt = gradecnt;
                stopshoot = 0;
            end
            else begin
                if (Block_X > Target_X && Block_X < Target_X + TARGET_WIDTH
- BLOCK_WIDTH
                &&  Block_Y > Target_Y && Block_Y < Target_Y + TARGET_HEIGHT
- BLOCK_HEIGHT
                && !stopshoot)
                begin
                    gradecnt = gradecnt + 1;
                    Target_X <= targetcount[17:9] + 50;
                    Target_Y <= targetcount[8:0] / 2 + 50;
                end
                else begin
                    gradecnt = gradecnt;
```

```
                Target_X <= Target_X;
                Target_Y <= Target_Y;
            end
            stopshoot = 1;
        end
    end
end

// 产生行时序
always @(posedge R_clk_25MHz or negedge rst) begin
    if(!rst)
        R_HS_count <=  12'd0;
    else if(R_HS_count == C_HS_LINE_PERIOD - 1)
        R_HS_count <=  12'd0;
    else
        R_HS_count <=  R_HS_count + 1'b1;
end

assign O_HS = (R_HS_count < C_HS_SYNC_PULSE) ? 1'b0 : 1'b1;

// 产生场时序
always@(posedge R_clk_25MHz or negedge rst) begin
    if(!rst)
        R_VS_count <= 12'd0;
    else if(R_VS_count == C_VS_FRAME_PERIOD - 1'b1)
        R_VS_count <= 12'd0;
    else if(R_HS_count == C_HS_LINE_PERIOD - 1'b1)
        R_VS_count <= R_VS_count + 1'b1;
    else
        R_VS_count <= R_VS_count;
end

assign O_VS = (R_VS_count < C_VS_SYNC_PULSE) ? 1'b0 : 1'b1;


assign Active_Flag  =   (R_HS_count >= (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH               )) &&
                        (R_HS_count < (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH + C_HS_ACTIVE_TIME))&&
                        (R_VS_count >= (C_VS_SYNC_PULSE +
C_VS_BACK_PORCH               )) &&
                        (R_VS_count < (C_VS_SYNC_PULSE +
C_VS_BACK_PORCH + C_VS_ACTIVE_TIME));
```

```
    // 把显示器屏幕分成 8 个纵列，每个纵列的宽度是 80
    always @(posedge R_clk_25MHz or negedge rst) begin
        if(!rst) begin
                O_RED   <= 4'b0000;
                O_GREEN <= 4'b0000;
                O_BLUE  <= 4'b0000;
                MAP_ADDR = 0;
        end
        else if(Active_Flag) begin
            case(live)
            1'b1: begin
                if (R_HS_count >= (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Gun_X)
                && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH + Gun_X
+ 192)
                && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Gun_Y)
                && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH + Gun_Y
+ 96))
                begin  //在 GUN 内
                    if (GUN_COLOR == 12'b111100001111) begin
                        if(R_HS_count >= (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH + Block_X)
                        && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Block_X + BLOCK_WIDTH)
                        && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Block_Y)
                        && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Block_Y + BLOCK_HEIGHT))
                        begin// 红色彩条
                            O_RED   <=  4'b1111;
                            O_GREEN <=  4'b0000;
                            O_BLUE  <=  4'b0000;
                        end
                        else if(R_HS_count >= (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH + Block_X)
                        && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Block_X + BLOCK_WIDTH))
                        begin
                            O_RED   <=  4'b0000;
                            O_GREEN <=  4'b0000;
                            O_BLUE  <=  4'b0000;
                        end
```

```
                    else if(R_VS_count >= (C_VS_SYNC_PULSE +
C_VS_BACK_PORCH + Block_Y)
                    && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Block_Y + BLOCK_HEIGHT))
                    begin
                        O_RED   <=  4'b0000;
                        O_GREEN <=  4'b0000;
                        O_BLUE  <=  4'b0000;
                    end
                    else if (R_HS_count >= (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH + Target_X)
                    && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Target_X + TARGET_WIDTH)
                    && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Target_Y)
                    && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Target_Y + TARGET_HEIGHT))
                    begin
                        if (MAN_COLOR == 12'b000011110000) begin
                            O_RED   <=  MAP_COLOR[11:8];
                            O_GREEN <=  MAP_COLOR[7:4];
                            O_BLUE  <=  MAP_COLOR[3:0];
                        end
                        else begin
                            O_RED   <=  MAN_COLOR[11:8];
                            O_GREEN <=  MAN_COLOR[7:4];
                            O_BLUE  <=  MAN_COLOR[3:0];
                        end
                    end
                    else begin
                        O_RED   <=  MAP_COLOR[11:8];
                        O_GREEN <=  MAP_COLOR[7:4];
                        O_BLUE  <=  MAP_COLOR[3:0];
                    end
                end
                else begin
                    O_RED   <=  GUN_COLOR[11:8];
                    O_GREEN <=  GUN_COLOR[7:4];
                    O_BLUE  <=  GUN_COLOR[3:0];
                end
            end
            else if(R_HS_count >= (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Target_X)
            && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
```

```
Target_X + TARGET_WIDTH)
              && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Target_Y)
              && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Target_Y + TARGET_HEIGHT))
              begin  //在 TARGET 内
                  if(R_HS_count >= (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Block_X)
                  && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Block_X + BLOCK_WIDTH)
                  && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y)
                  && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y + BLOCK_HEIGHT))
                  begin// 红色彩条
                      O_RED   <=  4'b1111;
                      O_GREEN <=  4'b0000;
                      O_BLUE  <=  4'b0000;
                  end
                  else if(R_HS_count >= (C_HS_SYNC_PULSE +
C_HS_BACK_PORCH + Block_X)
                  && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Block_X + BLOCK_WIDTH))
                  begin
                      O_RED   <=  4'b0000;
                      O_GREEN <=  4'b0000;
                      O_BLUE  <=  4'b0000;
                  end
                  else if(R_VS_count >= (C_VS_SYNC_PULSE +
C_VS_BACK_PORCH + Block_Y)
                  && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y + BLOCK_HEIGHT))
                  begin
                      O_RED   <=  4'b0000;
                      O_GREEN <=  4'b0000;
                      O_BLUE  <=  4'b0000;
                  end
                  else begin
                      if (MAN_COLOR == 12'b000011110000) begin
                          O_RED   <=  MAP_COLOR[11:8];
                          O_GREEN <=  MAP_COLOR[7:4];
                          O_BLUE  <=  MAP_COLOR[3:0];
                      end
                      else begin
```

```
                        O_RED   <=  MAN_COLOR[11:8];
                        O_GREEN <=  MAN_COLOR[7:4];
                        O_BLUE  <=  MAN_COLOR[3:0];
                    end
                end

            end
            else if(R_HS_count >= (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Block_X)
            && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Block_X + BLOCK_WIDTH)
            && R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y)
            && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y + BLOCK_HEIGHT))
            begin// 红色彩条
                O_RED   <=  4'b1111;
                O_GREEN <=  4'b0000;
                O_BLUE  <=  4'b0000;
            end
            else if(R_HS_count >= (C_HS_SYNC_PULSE + C_HS_BACK_PORCH
+ Block_X)
            && R_HS_count < (C_HS_SYNC_PULSE + C_HS_BACK_PORCH +
Block_X + BLOCK_WIDTH))
            begin
                O_RED   <=  4'b0000;
                O_GREEN <=  4'b0000;
                O_BLUE  <=  4'b0000;
            end
            else if(R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH
+ Block_Y)
            && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH +
Block_Y + BLOCK_HEIGHT))
            begin
                O_RED   <=  4'b0000;
                O_GREEN <=  4'b0000;
                O_BLUE  <=  4'b0000;
            end
            else begin
                O_RED   <=  MAP_COLOR[11:8];
                O_GREEN <=  MAP_COLOR[7:4];
                O_BLUE  <=  MAP_COLOR[3:0];
            end
            if (MAP_ADDR == MAP_MAXCOUNT - 1)
```

```
                    MAP_ADDR = 0;
                else
                    MAP_ADDR = MAP_ADDR + 1;
            end
            1'b0: begin
                if(R_VS_count >= (C_VS_SYNC_PULSE + C_VS_BACK_PORCH + 200)
                && R_VS_count < (C_VS_SYNC_PULSE + C_VS_BACK_PORCH + 280))
                begin
                    O_RED   <=  OVER_COLOR[11:8];
                    O_GREEN <=  OVER_COLOR[7:4];
                    O_BLUE  <=  OVER_COLOR[3:0];
                end
                else begin
                    O_RED   <=  4'b0000;
                    O_GREEN <=  4'b0000;
                    O_BLUE  <=  4'b0000;
                end
            end
            endcase
        end
        else begin
            O_RED   <=  4'b0000;
            O_GREEN <=  4'b0000;
            O_BLUE  <=  4'b0000;
            MAP_ADDR = MAP_ADDR;
        end
    end
endmodule
```

五、测试模块建模

对若干子模块进行测试。
test bench 代码如下

1. *timecnt 子模块*

此处为了使测试更加直观、简单，设定测试时钟远低于实际的 100MHz，同时倒计时时间也调整为 10s，便于观察

```
`timescale 1ns / 1ns
module test_tb();
    reg clk;
    reg rst;
```

```
    wire live;
    wire [15:0] Time;
    always begin
        clk = 0;
        #1;
        clk = 1;
        #1;
    end
    initial begin
        rst = 0;
        #10;
        rst = 1;
    end
    timecnt t(clk, rst, live, Time);
endmodule
```

2. *L3G4200D 子模块*

由于未连接部件，只是测试，SDO 无法测试，除此之外的信号都能正常测试。

```
`timescale 1ns / 1ps
module test_tb();
    reg clk;
    reg rst;
    reg sdo;
    wire cs;
    wire spc;
    wire sdi;
    wire [15:0]X_data;
    wire [15:0]Y_data;

    always begin
        clk = 0;
        #0.5;
        clk = 1;
        #0.5;
    end
    initial begin
        rst = 0;
        sdo = 0;
        #5;
        rst = 1;
    end

    L3G4200D l3g4200d(clk, rst, sdo, cs, spc, sdi, X_data, Y_data);
endmodule
```

3. *SegmentDisplay 子模块*

```
`timescale 1ns / 1ns
module test_tb();
    reg clk;
    reg rst;
    reg [15:0]score;
    reg [15:0]Time;
    wire [6:0]oData;
    wire [7:0]AN;

    always begin
        clk = 0;
        #1;
        clk = 1;
        #1;
    end
    always begin
        Time = Time - 1;
        #100;
    end
    initial begin
        rst = 0;
        score = 0;
        Time = 10;
        #10;
        rst = 1;
        #230;
        score = score + 1;
        #80;
        score = score + 1;
        #120;
        score = score + 1;
        #310
        score = score + 1;
    end

    SegmentDisplay display(clk, rst, Score, Time, oData, AN);
endmodule
```

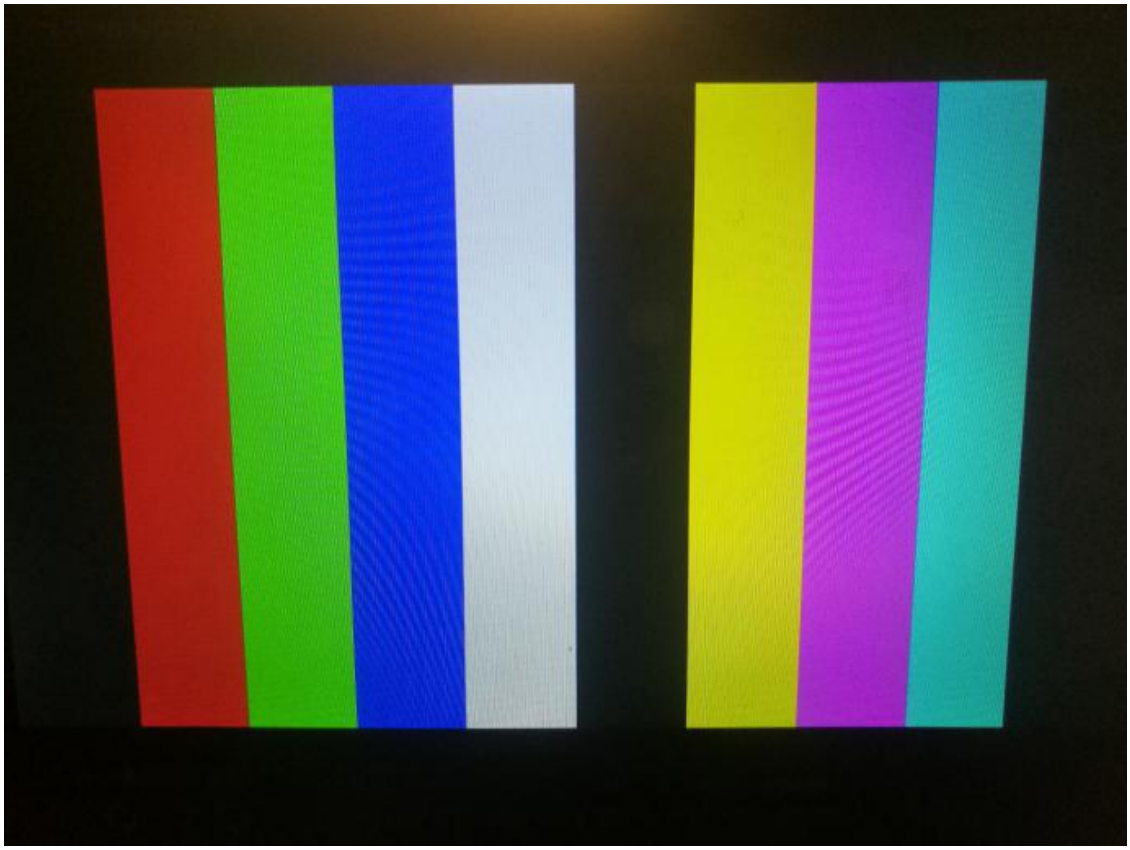
4. *BlockMove 子模块*

```
`timescale 1ns / 1ns
module test_tb();
    reg clk;
    reg rst;
    reg setmid;
    reg [15:0]X_data;
    reg [15:0]Y_data;
    wire [9:0]X_angular;
    wire [9:0]Y_angular;

    always begin
        clk = 0;
        #1;
        clk = 1;
        #1;
    end
    initial begin
        rst = 0;
        setmid = 0;
        X_data = 16'b10000000_00000000;
        Y_data = 16'b10000000_00000000;
        #10;
        rst = 1;
        #90;
        X_data = 16'b11101100_11011100;
        Y_data = 16'b10000100_11011100;
        #100;
        X_data = 16'b110010010_01011100;
        Y_data = 16'b10100000_11011110;
        #100;
        X_data = 16'b00110000_10010100;
        Y_data = 16'b01100100_11011001;
        #50;
        setmid = 1;
        #50;
        X_data = 16'b00000000_11011100;
        Y_data = 16'b01110100_11011100;
        #100;
        setmid = 0;
        #100;
        X_data = 16'b01000011_11011100;
        Y_data = 16'b01110100_00011101;
```

```
        #100;
        X_data = 16'b10110000_01010100;
        Y_data = 16'b10100001_11101100;
        #50;
        setmid = 1;
        #50;
        rst = 0;
    end

    BlockMove bm(clk, rst, setmid, X_data, Y_data, X_angular, Y_angular);
endmodule
```

5. *VGA 子模块*

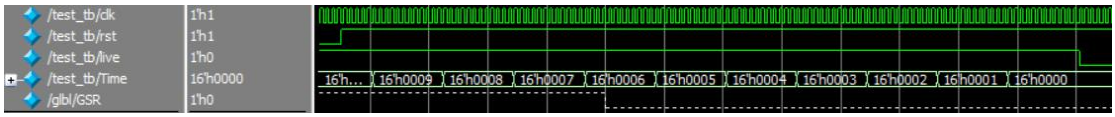
由于 VGA 模块显示的特殊性，以及射击键需对准目标等因素，直接下板测试更为便捷，因此该模块暂不使用 test bench 测试，直接使用最简单的 8 色彩条测试 VGA 是否能正常工作，经测试，预期中的 8 色彩条能够正常显示，因此 VGA 模块逻辑正确。



六、实验结果

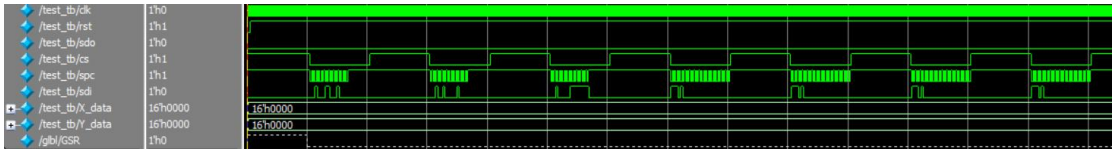
1. *timecnt 子模块*

如图所示为 ModelSim 波形图，顺利完成预期倒计时功能。



2. *L3G4200D 子模块*

如图所示，CS 时序正确；前三个周期的 SPI 波形为写寄存器指令，后面均为读取，SPC 呈现出预期的时钟效果。结果正确。

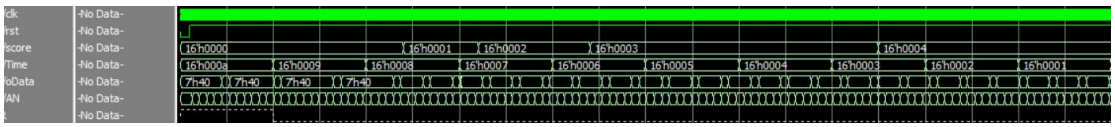


3. *SegmentDisplay 子模块*

数码管显示数字与 oData 输出有如下对应关系

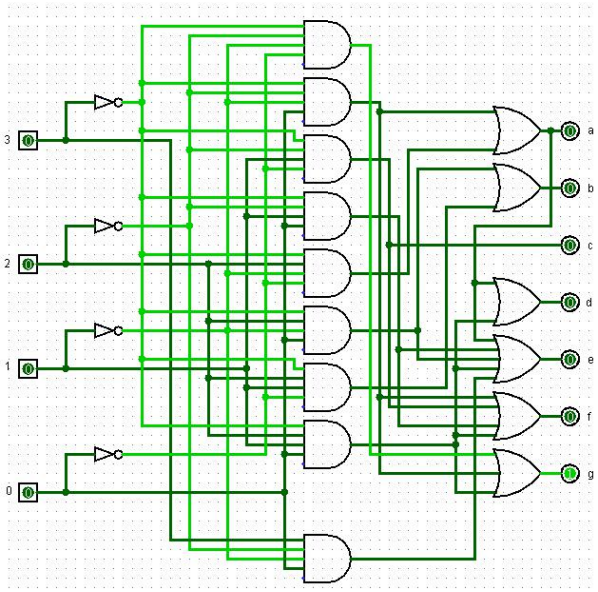
显示的数字	十六进制 oData	显示的数字	十六进制 oData
0	7’h40	5	7’h12
1	7’h79	6	7’h02
2	7’h24	7	7’h78
3	7’h30	8	7’h00
4	7’h19	9	7’h10

ModelSim 波形图如下：



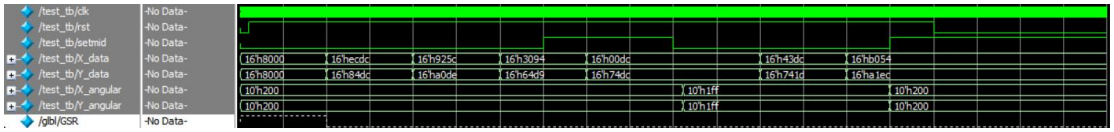
如图所示为 ModelSim 波形图，放大查看后发现数码管能够正常显示实时的 Time 与 score 数据。

Logisim 贴图：



直接参考之前小作业 display7 模块，是输入数据转为 oData 的电路。经验证，与预期目标一一对应。

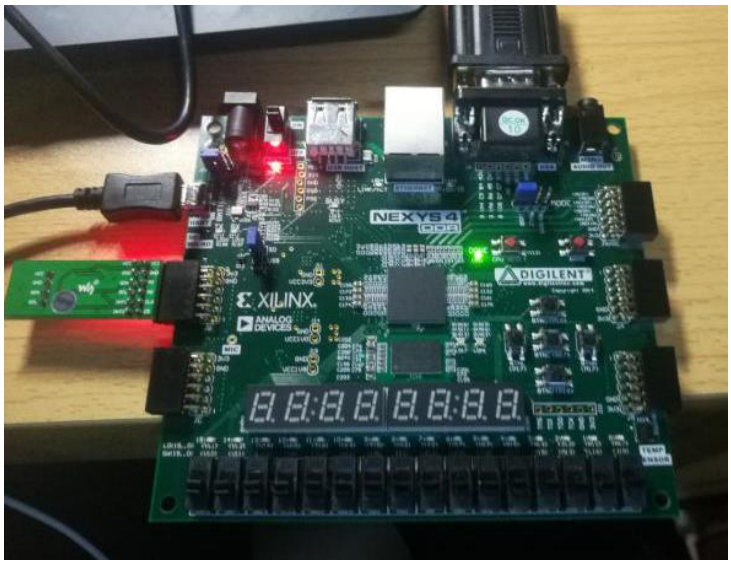
4. *BlockMove 子模块*

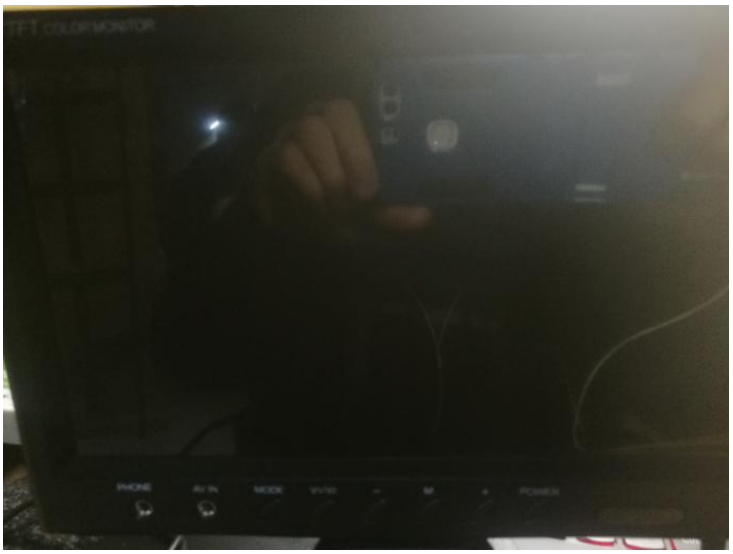


观察波形，发现 angular 改变较为困难，因为为了精确模拟积分过程，angular 结果是在高频时钟下累加产生的，Simulation 中时钟频率较低。经过观察，setmid 和 angular 还是能够向预期方向改变的。

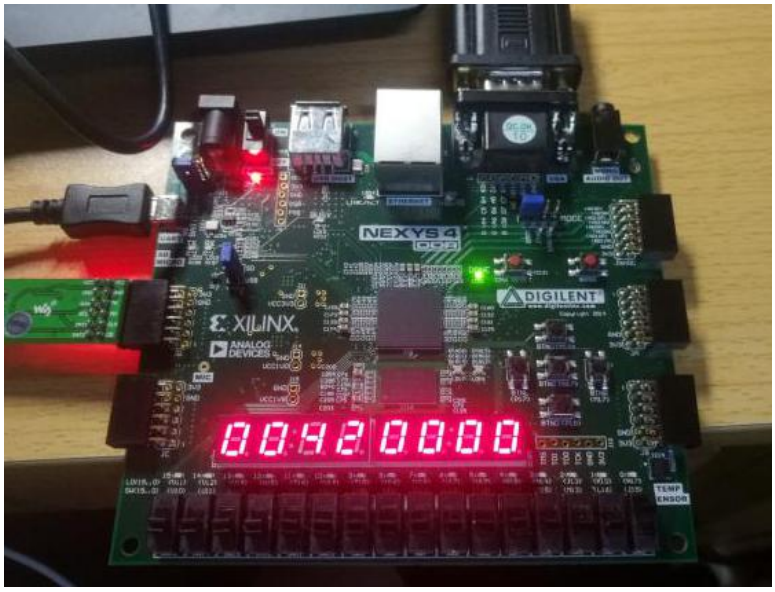
总下板贴图：

①如图所示，最初还未打开 rst 游戏开关，板上及屏幕无任何显示；





②打开开关后，屏幕显示画面，板子上开始计分及倒计时；



③转动板子，对准敌人，按下射击键，完成一次得分；

可以看到计分由 1 变为 2，在屏幕随机位置新出现一个敌人。

④倒计时结束，屏幕显示 Game Over。

七、结论

各个子模块无论是单独验证，还是合在一起下板，均能按照预期正常工作。因此这一数字系统是能够正常工作的。

但是对于 L3G4200D 的最优读取速率，以及对其更加强大的高通、低通滤波器，Interrupt 功能等的使用仍然可以进一步探索，获取到更加优质的数据。

八、心得体会及建议

处理一个规模较大的数字系统时，应当合理地对其进行分割，使之成为若干个规模较小的子系统，使问题简化。

在调试过程中，发现当一个子模块的逻辑有错误、混乱的时候，甚至在硬件层面会影响到另一个与之毫不相干的子模块，以至于整体调试时，不知道是哪个模块产生问题。因此在编写时，可以先分别编写子模块、单独测试各个子模块是否功能完善。待确保所有子模块功能完善时，再拼接成一个完整的系统，进行调试。

另外，由于最初想法只是用陀螺仪做一个小游戏。最终做出一个 CS 射击小游戏时，才发现开枪时没有声音是美中不足的一个地方，但是此时已经没有更多的时间去申领新的部件甚至去研究、优化升级了。在最初时对自己要做的仍然不够明确，因此最终留下一个不大不小的遗憾。

本次大作业，最初寻找相关资料花费了大量的时间。其中 L3G4200D 三轴陀螺仪的 data sheet 所提供的信息严重不足，甚至连部件内部的寄存器地址、各个寄存器的作用简介都没提及，更给前期摸索增加了极大的困难。而且网络上所有关于这一部件的参考代码均为 C 语言编写，且所调用的库函数均超出大二学生所掌握的知识。

最终我还是通过谷歌，找到了一款名为 L3GD20 的三轴陀螺仪，并发现这一款陀螺仪和 L3G4200D 属于同一系列，差别极小。在意法半导体公司的官网上寻找到 L3GD20 的更加齐全的全英文相关资料后，最终摸索出使用 SPI 协议激活 L3G4200D 并对其读写的方法，这段工作花费了我一半的时间及经历，实属不易。