同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2151127

姓　　名 华洲琦

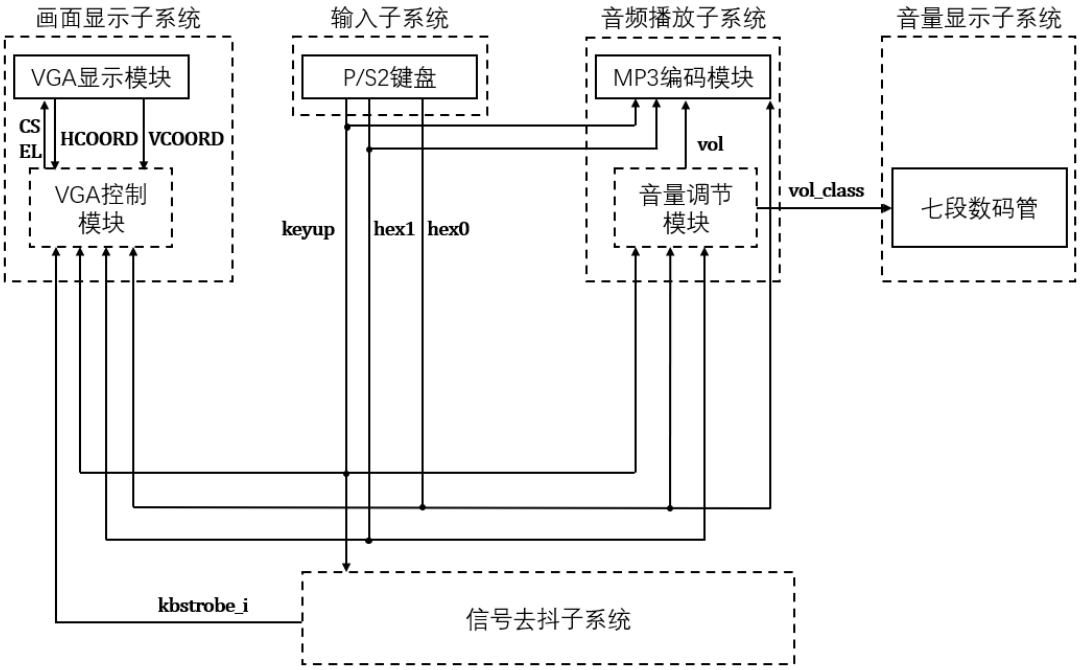
专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

基于 VGA、MP3 和 PS2 键盘实现、面向幼年群体、具有较强拓展性的迷宫闯关益智游戏系统。

二、迷宫游戏数字系统总框图



画面显示子系统：由 VGA 显示模块和 VGA 控制模块组成，VGA 显示模块扫描行列值并将坐标 HCOORD 和 VCOORD 传给 VGA 控制模块，控制模块运算后将颜色值 CSEL 回传给显示模块，由显示模块的 output 接口连接 VGA 硬件管脚进行颜色数据传输和显示。同时 VGA 控制模块接收去抖后的通断信息 kbstrobe_i、键盘弹起信号 keyup、键码 hex1 和 hex0，并改变小球位置，通过 VGA 显示模块显示小球。

输入子系统：由 P/S2 键盘组成，输出有效键码 hex1 和 hex0 和弹起信号 keyup（按键后弹起则 keyup 为 1），提供给画面显示子系统和音频播放子系统，以实现按下按键时小球的位置移动和提示音效的播放。

音频播放子系统：由 MP3 编码模块和音量调节模块组成，一方面可以通过键盘控制音量调节模块改变音量大小、将音量数值传给 MP3 编码模块，也可以读取键盘控制小球的移动键（W/A/S/D）并发出不同提示音效。

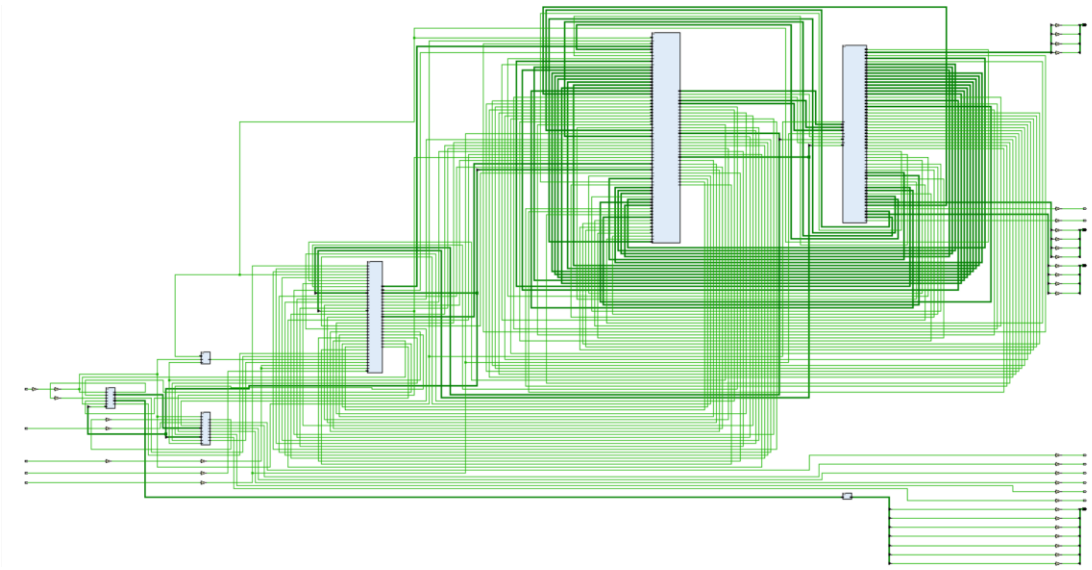
音量显示子系统：由七段数码管组成，同步接收音量调节模块中实时的音量级别并用数码

管显示。

信号去抖子系统：接收 keyup 值，输出去抖后的通断信号 kbstrobe_i，确保键盘控制小球移动的稳定性，避免时序逻辑中抖动造成的尖峰脉冲信号

三、RTL 分析

项目综合后原理图：



四、子系统模块建模

4.1 键盘控制模块

键盘模块按照 PS2 协议（双向同步串行通讯协议）传输数据，获取键盘扫描码，按下发发送断码，弹起发送通码，本模块通过检测断码中的 0x0F 获取数据。

使用 22 位的移位寄存器存储，当寄存器中存储的值为“0FXX”（XX 表示两个 16 进制键码，分别对应接口中的 HEX0 和 HEX1）时表示键盘被按下。控制信号 ARST_L 低电平时将寄存器清零。因此只需判断移位寄存器存储的前 8 位是否位 0X，即可判断发送控制信号 KEYUP 是否有效。

需要注意的是，作为时序逻辑的常见问题，在使用时钟周期读取按下的键码时会出现“抖动信号”，具体表现为单次按下某按键，重复多次操作甚至出现其他不可预知操作。因此需要额外设计模块进行键码读取的防抖动操作。（详见“防抖动模块”）

此外，键盘模块使用的时钟是同步 usb 时钟，不同于 N4 板 e3 管脚的系统时钟。

接口定义：

```
module Keyboard (
```

```
    input CLK,      //同步后 usb 信号
    input SDATA,    //同步后 usb 数据
    input ARST_L,   //控制信号
    output [3:0] HEX0,  //16 进制键码
    output [3:0] HEX1,  //16 进制键码
    output reg KEYUP    //返回是否读取到键的信息，1 为读取到
    );
```

模块实现：

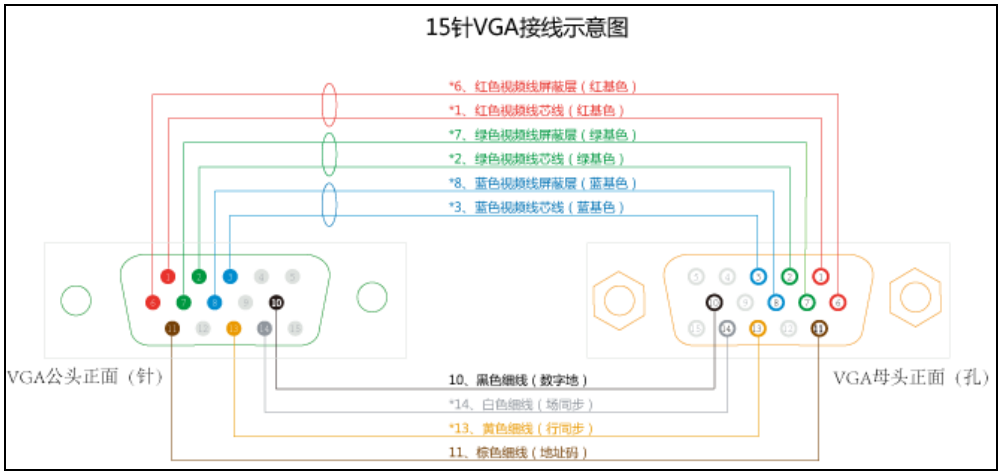
```
module Keyboard (
    input CLK,      //同步后 usb 信号
    input SDATA,    //同步后 usb 数据
    input ARST_L,   //控制信号
    output [3:0] HEX0,  //16 进制键码
    output [3:0] HEX1,  //16 进制键码
    output reg KEYUP    //返回是否读取到键的信息，即传达了一个键盘有效信息
    );
wire arst_i, rollover_i;
reg [21:0] Shift;       //移位寄存器
assign arst_i = ~ARST_L;    //更新控制信号
assign HEX0[3:0] = Shift[15:12];
assign HEX1[3:0] = Shift[19:16];
always @(negedge CLK or posedge arst_i) begin;
    if(arst_i)begin
        Shift <= 22'b0000000000000000000000;
    end
    else begin
        Shift <= {SDATA, Shift[21:1]}; //从最后一位移入
    end
end
always @(posedge CLK) begin
    if(Shift[8:1] == 8'hF0) begin
        KEYUP <= 1'b1;
    end
    else begin
        KEYUP <= 1'b0;
    end
end
endmodule
```

4.2 VGA 显示模块

VGA（Video Graphics Array）显示设备通过 VGA 电缆和 N4 开发板相连，主要传输 RGBHV 五种信号：

R	红色	对应端口 RED
G	绿色	对应端口 GREEN
B	蓝色	对应端口 BLUE
H	行同步信号	对应端口 HSYNC
V	场同步信号	对应端口 VSYNC

对应光缆接口：



使用同步开关驱动不同像素点颜色的绘制。由于 VGA 必须在 25mhz 时钟频率下工作，需要将系统时钟 CLK 进行 4 分频得到用于信号传输的 CLKOUT 信号，通过同步状态下改变 CSEL 可以切换在该像素点绘制的颜色。

见接口定义，七个 output 信号中 HSYNC / VSYNC / RED / GREEN / BLUE 传入 VGA 电缆，HCOORD / VCOORD 获取同步实时的坐标信号，经过 VGA 控制模块（详见 3.3）判断后给出 R/G/B 值传入光缆，实现颜色绘制。

接口定义：

```
module VGA(
    input CLK,                  //系统时钟
    input [11:0] CSEL,          //颜色 r,g,b 值
    input ARST_L,               //控制信号
    output HSYNC,               //行同步信号
    output VSYNC,               //场同步信号
    output reg [3:0] RED,
    output reg [3:0] GREEN,
```

```
    output reg [3:0] BLUE,
    output reg [9:0] HCOORD,    //横坐标
    output reg [9:0] VCOORD     //纵坐标
    );
```

模块建模：

```
module VGA(
    input CLK,              //系统时钟
    input [11:0] CSEL,      //颜色 r,g,b 值
    input ARST_L,           //控制信号
    output HSYNC,           //行同步信号
    output VSYNC,           //场同步信号
    output reg [3:0] RED,
    output reg [3:0] GREEN,
    output reg [3:0] BLUE,
    output reg [9:0] HCOORD,    //横坐标
    output reg [9:0] VCOORD     //纵坐标
    );
wire aclr_i;
wire Hrollover_i, Vrollover_i;  //翻转信号
assign aclr_i = ~ARST_L;    //反向控制信号
reg SREG;
reg CLKOUT;
//当水平达到最右端或者垂直到达最下端时信号翻转（800,525）
//异步翻转，故可以到达临界值
assign Hrollover_i = (HCOORD[9] & HCOORD[8] & HCOORD[5]) ? 1'b1 : 1'b0;
//800 D = 001100100000 B
assign Vrollover_i = (VCOORD[9] & VCOORD[3] & VCOORD[2] & VCOORD[0]) ?
1'b1 : 1'b0;     //525 D = 001000001101 B
/* part1 坐标控制 */
//HCOORD 在每个时钟周期计数或重置为 0
always @(posedge CLKOUT or posedge aclr_i) begin
    if(aclr_i) begin
        HCOORD <= 10'b0000000000;   //控制信号
    end
    else if(Hrollover_i)
        HCOORD <= 10'b0000000000;   //横向翻转信号有效
    else
        HCOORD <= HCOORD + 1;
```

```
end
//每次 Hrollover_i 被声明或重置为 0 时，HCOORD 都会计数
always @(posedge CLKOUT or posedge aclr_i) begin
    if(aclr_i) begin
        VCOORD <= 10'b0000000000;
    end
    else if(Vrollover_i)
        VCOORD <= 10'b0000000000;
    else if(Hrollover_i)
        VCOORD <= VCOORD + 1;
end
assign HSYNC = ((HCOORD < 756) && (HCOORD > 658)) ? 1'b0 : 1'b1;
assign VSYNC = ((VCOORD < 495) && (VCOORD > 492)) ? 1'b0 : 1'b1;
/* part 3 颜色绘制 */
always @(posedge CLKOUT or posedge aclr_i) begin
    if (aclr_i) begin
        RED = 4'h0;
        GREEN = 4'h0;
        BLUE = 4'h0;
    end
    else if((HCOORD > 640) || (VCOORD > 480)) begin
        RED = 4'h0;
        GREEN = 4'h0;
        BLUE = 4'h0;
    end
    else if((HCOORD < 320) || (VCOORD < 240)) begin
        RED = 4'h0;
        GREEN = 4'hF;
        BLUE = 4'h0;
    end
    else begin
        // rgb 值由 vgacontroller 决定
        RED <= CSEL[11:8];
        GREEN <= CSEL[7:4];
        BLUE <= CSEL[3:0];
    end
end
/* part 4 时钟分频 */
//25mhz 运行（否则报错信号格式无效）
```

```
//2 分频
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        SREG <= 1'b0;
    end
    else begin
        SREG <= ~SREG;
    end
end
//4 分频
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        CLKOUT <= 1'b0;
    end
    else if(SREG) begin
        CLKOUT <= ~CLKOUT;
    end
end
endmodule
```

4.3 VGA 控制模块

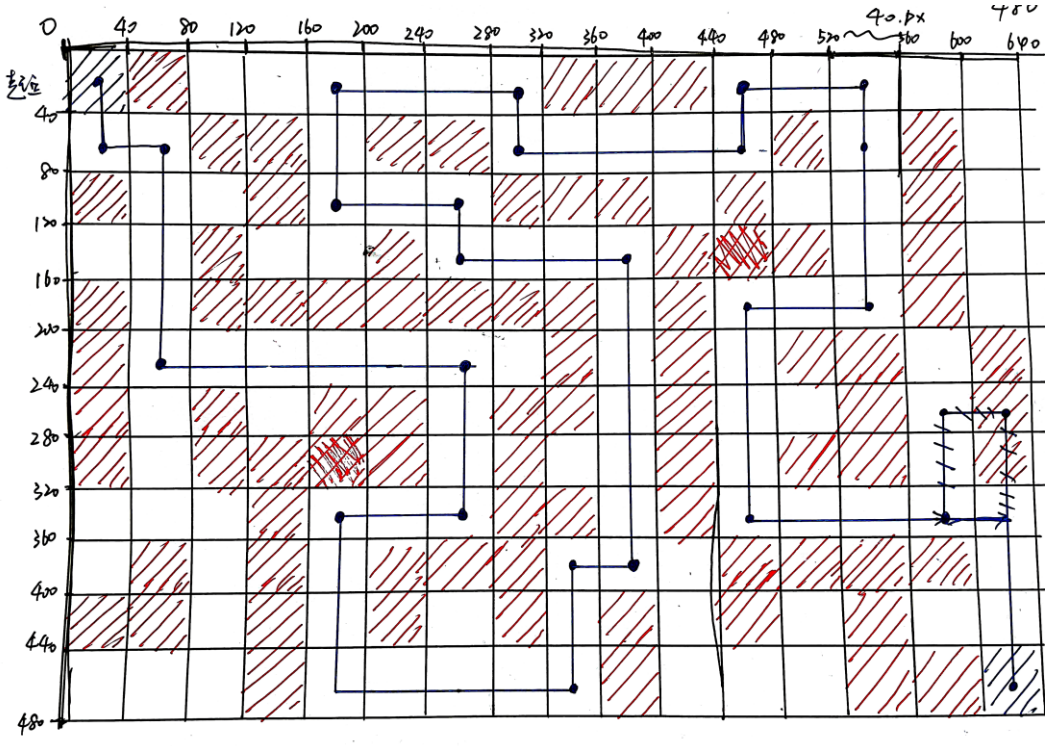
VGA 控制模块和 VGA 显示模块使用同步时钟信号，主要有三个功能：

1. 实时接收 VGA 显示模块中正在扫描的行列坐标（VCOORD / HCOORD），经过判断，传送 RGB 颜色值到 VGA 显示模块进行颜色绘制；
2. 读入键码信息，接受当前小球坐标（ballX / ballY），判断是否进行小球的移动，更新小球坐标（ballX_now）；
3. 游戏地图存储（Map 二维数组）

关于该游戏的地图，将整个 VGA 显示部分（640px*480px）分为 40px 边长的小正方块，即水平方向 16 块，纵向 12 块。用这些正方形块构成迷宫走道、起点终点和墙壁，在 initial 块中对 Map[0:11][0:15]初始化静态存储。存储方式和对应颜色如下表：

组成部分	**Map 存储值**	**显示颜色（RGB）**
墙壁	1	**条纹** 深黄：12’b 1110_1011_0000 浅黄：12’b 1110_0111_0000
走道	2	12’b 0000_0000_0000
起点终点	0	12’b 0000_1100_1110

地图模型如下：



下面简单介绍其他细节刻画：

- **小球绘制**：小球是半径为 13 的红色圆形，当前 VGA 绘制原理为获取行列坐标判断后回传颜色信息，故先确定小球所在 40*40 块坐标（*row, col*），计算正方形块的几何中心坐标（*(2*row+1)*20, (2*col)*20*），再判断当前像素点和集合中心距离是否小于 13，两种情况分别给出不同 RGB 信息。
- **表情绘制**：在小球绘制的基础上，内部绘制表情。由于是像素级刻画，经过多次实践后，得出表情坐标如下表：

表情组成	**像素范围**
左眼	VCOORD>=row*40+14 && VCOORD<=row*40+15 && HCOORD>=col*40+12 && HCOORD<=col*40+17
右眼	VCOORD>=row*40+14 && VCOORD<=row*40+15 && HCOORD>=col*40+23 && HCOORD<=col*40+28
嘴	VCOORD>=row*40+24 && VCOORD<=row*40+25 && HCOORD>=col*40+15 && HCOORD<=col*40+25

- **条纹绘制**：为了避免色块单调性，提高整体美观度，墙壁部分使用 8px 斜条纹绘制，其中斜条纹通过判断(VCOORD+HCOORD)/8 值是否为偶数绘制。

```
else if(Map[row][col]==1) begin      // 墙体
    // 花纹效果
    if(((VCOORD+HCOORD)/8)%2==0) CSEL<=12'b1110_1011_0000;
    else CSEL<=12'b1110_0111_0000;
```

- **移动分析**：模块通过输入 KBCODE 获取键盘的键值，如果为 W/S/A/D（对应上/下/左/右）则需要更新小球最新位置，同时需要判断小球是否移动到墙壁上，或边界外，以及是否到终点。其中考虑到 Verilog 语言不具有短路运算特性，有时不能同时判断多个条件，判断语句较为繁琐。
- **强可拓展性**：可拓展性体现在后期可以更方便的新增其他设备或模块，本模块将当前

小球横纵坐标 ballX_now 和 ballY_now 传入顶层模块，而不是存储在模块内，可以更方便的让后续可能需要的新增模块使用数据。

接口定义：

```
module VGA_controller(
    input CLK,              //系统时钟
    input kbstrobe_i,       //去抖信号
    input [7:0] KBCODE,     //按下键的键值
    input [9:0] HCOORD,     //当前横坐标
    input [9:0] VCOORD,     //当前纵坐标
    input ARST_L,           //控制信号
    input [5:0] ballX,
    input [5:0] ballY,
    output reg [5:0] ballX_now,
    output reg [5:0] ballY_now,
    output reg [11:0] CSEL,  //rgb 值
    output reg click
    );
```

模块建模：

```
module VGA_controller(
    input CLK,        //系统时钟
    input kbstrobe_i, //去抖信号
    input [7:0] KBCODE,//按下键值
    input [9:0] HCOORD,//当前横坐标
    input [9:0] VCOORD,//当前纵坐标
    input ARST_L,      //控制信号
    input [5:0] ballX,
    input [5:0] ballY,
    output reg [5:0] ballX_now,
    output reg [5:0] ballY_now,
    output reg [11:0] CSEL, //rgb
    output reg click
    );

    wire aclr_i;
    reg [7:0] Map[0:11][0:15];       // 1=墙体，0=走道，2=起点和终点
    reg SREG, CLKOUT;               //用于时钟分频
    wire [4:0] row,col;             //当前绘制的像素所在的区域坐标（40px 单
```

位正方形块）

```
    //reg [4:0] ballX,ballY;           //当前小球所在位置（坐标）
    //wire CLK_DET;
    parameter x_length = 15;     //地图右端点
    parameter y_height = 11;     //地图下端点
    assign aclr_i = ~ARST_L;    //反向控制信号
    assign row = (VCOORD<40?0:(VCOORD-40<40?1:(VCOORD-80<40?2:(VCOORD-
120<40?3:(VCOORD-160<40?4:(VCOORD-200<40?5:(VCOORD-240<40?6:(VCOORD-
280<40?7:(VCOORD-320<40?8:(VCOORD-360<40?9:(VCOORD-400<40?10:(VCOORD-
440<40?11:(VCOORD-480<40?12:(VCOORD-520<40?13:(VCOORD-
560<40?14:(VCOORD-600<40?15:0)))))))))))));
    //当前像素所在 Map 中的行坐标
    assign col = (HCOORD<40?0:(HCOORD-40<40?1:(HCOORD-80<40?2:(HCOORD-
120<40?3:(HCOORD-160<40?4:(HCOORD-200<40?5:(HCOORD-240<40?6:(HCOORD-
280<40?7:(HCOORD-320<40?8:(HCOORD-360<40?9:(HCOORD-400<40?10:(HCOORD-
440<40?11:(HCOORD-480<40?12:(HCOORD-520<40?13:(HCOORD-
560<40?14:(HCOORD-600<40?15:0)))))))))))));
    //当前像素所在 Map 中的列坐标
    initial begin   //迷宫地图（走道和墙体使用二位数字定位）
       Map[0][0]=2; Map[0][1]=1; Map[0][2]=0; Map[0][3]=0;
       Map[0][4]=0; Map[0][5]=0; Map[0][6]=0; Map[0][7]=0;
       Map[0][8]=1; Map[0][9]=1; Map[0][10]=1; Map[0][11]=0;
       Map[0][12]=0; Map[0][13]=0; Map[0][14]=0; Map[0][15]=0;
       Map[1][0]=0; Map[1][1]=0; Map[1][2]=1; Map[1][3]=1;
       Map[1][4]=0; Map[1][5]=1; Map[1][6]=1; Map[1][7]=0;
       Map[1][8]=0; Map[1][9]=0; Map[1][10]=0; Map[1][11]=0;
       Map[1][12]=1; Map[1][13]=0; Map[1][14]=1; Map[1][15]=0;
       Map[2][0]=1; Map[2][1]=0; Map[2][2]=0; Map[2][3]=1;
       Map[2][4]=0; Map[2][5]=0; Map[2][6]=0; Map[2][7]=1;
       Map[2][8]=1; Map[2][9]=1; Map[2][10]=0; Map[2][11]=1;
       Map[2][12]=0; Map[2][13]=0; Map[2][14]=1; Map[2][15]=0;
       Map[3][0]=0; Map[3][1]=0; Map[3][2]=1; Map[3][3]=0;
       Map[3][4]=0; Map[3][5]=1; Map[3][6]=0; Map[3][7]=0;
       Map[3][8]=0; Map[3][9]=0; Map[3][10]=1; Map[3][11]=1;
       Map[3][12]=1; Map[3][13]=0; Map[3][14]=1; Map[3][15]=0;
       Map[4][0]=1; Map[4][1]=0; Map[4][2]=1; Map[4][3]=1;
       Map[4][4]=1; Map[4][5]=0; Map[4][6]=1; Map[4][7]=1;
       Map[4][8]=1; Map[4][9]=0; Map[4][10]=1; Map[4][11]=0;
       Map[4][12]=0; Map[4][13]=0; Map[4][14]=1; Map[4][15]=0;
```

```
        Map[5][0]=1; Map[5][1]=0; Map[5][2]=0; Map[5][3]=0;
        Map[5][4]=0; Map[5][5]=0; Map[5][6]=0; Map[5][7]=0;
        Map[5][8]=1; Map[5][9]=0; Map[5][10]=1; Map[5][11]=0;
        Map[5][12]=1; Map[5][13]=1; Map[5][14]=0; Map[5][15]=1;
        Map[6][0]=1; Map[6][1]=0; Map[6][2]=1; Map[6][3]=0;
        Map[6][4]=1; Map[6][5]=1; Map[6][6]=0; Map[6][7]=1;
        Map[6][8]=1; Map[6][9]=0; Map[6][10]=1; Map[6][11]=0;
        Map[6][12]=0; Map[6][13]=1; Map[6][14]=0; Map[6][15]=1;
        Map[7][0]=1; Map[7][1]=0; Map[7][2]=1; Map[7][3]=1;
        Map[7][4]=0; Map[7][5]=1; Map[7][6]=0; Map[7][7]=1;
        Map[7][8]=0; Map[7][9]=0; Map[7][10]=1; Map[7][11]=0;
        Map[7][12]=1; Map[7][13]=1; Map[7][14]=0; Map[7][15]=1;
        Map[8][0]=0; Map[8][1]=0; Map[8][2]=0; Map[8][3]=1;
        Map[8][4]=0; Map[8][5]=0; Map[8][6]=0; Map[8][7]=0;
        Map[8][8]=1; Map[8][9]=0; Map[8][10]=1; Map[8][11]=0;
        Map[8][12]=0; Map[8][13]=0; Map[8][14]=0; Map[8][15]=0;
        Map[9][0]=0; Map[9][1]=1; Map[9][2]=0; Map[9][3]=1;
        Map[9][4]=0; Map[9][5]=1; Map[9][6]=1; Map[9][7]=1;
        Map[9][8]=0; Map[9][9]=0; Map[9][10]=0; Map[9][11]=1;
        Map[9][12]=1; Map[9][13]=1; Map[9][14]=1; Map[9][15]=0;
        Map[10][0]=1; Map[10][1]=1; Map[10][2]=0; Map[10][3]=1;
        Map[10][4]=0; Map[10][5]=1; Map[10][6]=0; Map[10][7]=1;
        Map[10][8]=0; Map[10][9]=1; Map[10][10]=0; Map[10][11]=1;
        Map[10][12]=0; Map[10][13]=1; Map[10][14]=0; Map[10][15]=0;
        Map[11][0]=0; Map[11][1]=0; Map[11][2]=0; Map[11][3]=1;
        Map[11][4]=0; Map[11][5]=0; Map[11][6]=0; Map[11][7]=0;
        Map[11][8]=0; Map[11][9]=1; Map[11][10]=0; Map[11][11]=0;
        Map[11][12]=0; Map[11][13]=1; Map[11][14]=1; Map[11][15]=2;
    end
    //绘制迷宫地图
    //根据横纵坐标判断是否绘制墙体/走道/起点终点
    always @(posedge CLKOUT) begin
        if(VCOORD>=0&&VCOORD<=480&&HCOORD>=0&&HCOORD<=640) begin
            if(Map[row][col]==0) begin          // 走道
                if(row==ballX&&col==ballY&&((HCOORD-
(2*col+1)*20)*(HCOORD-(2*col+1)*20)+(VCOORD-(2*row+1)*20)*(VCOORD-
(2*row+1)*20)<13*13)) begin
                    if(VCOORD>=row*40+14&&      VCOORD<=row*40+15      &&
HCOORD>=col*40+12 && HCOORD<=col*40+17)     //左眼
```

```
                        CSEL<=12'hFFF;
                    else  if(VCOORD>=row*40+14&&  VCOORD<=row*40+15  &&
HCOORD>=col*40+23 && HCOORD<=col*40+28)//右眼
                        CSEL<=12'hFFF;
                    else  if(VCOORD>=row*40+24&&  VCOORD<=row*40+25  &&
HCOORD>=col*40+15 && HCOORD<=col*40+25)
                        CSEL<=12'hFFF;
                    else
                        CSEL<=12'hF00;              // 小球
                end
                else CSEL<=12'h0F0;
            end
            else if(Map[row][col]==1) begin // 墙体
       // 花纹效果
       if(((VCOORD+HCOORD)/8)%2==0) CSEL<=12'b1110_1011_0000;
                else CSEL<=12'b1110_0111_0000;
            end
            else     if(Map[row][col]==2)begin     //    起    终    点
if(row==ballX&&col==ballY&&((HCOORD-(2*col+1)*20)*(HCOORD-
(2*col+1)*20)+(VCOORD-(2*row+1)*20)*(VCOORD-(2*row+1)*20)<13*13)) begin
if(VCOORD>=row*40+14&&   VCOORD<=row*40+15   &&   HCOORD>=col*40+12   &&
HCOORD<=col*40+17)     //左眼
                        CSEL<=12'hFFF;
                    else  if(VCOORD>=row*40+14&&  VCOORD<=row*40+15  &&
HCOORD>=col*40+23 && HCOORD<=col*40+28)//右眼
                        CSEL<=12'hFFF;
                    else  if(VCOORD>=row*40+24&&  VCOORD<=row*40+25  &&
HCOORD>=col*40+15 && HCOORD<=col*40+25)
            CSEL<=12'hFFF;
                    else
       CSEL<=12'hF00;      // 小球
                end
                else CSEL<=12'b0000_1100_1110;
            end
        end
    end
    //根据获取到的键码判断移动方向
    always @(posedge CLKOUT or posedge aclr_i) begin
        if(aclr_i) begin
```

```
            ballX_now <= 0;
            ballY_now <= 0;
            click<=0;
        end
        else if (kbstrobe_i) begin
                case(KBCODE)
                    // 根据输入的键码确定位置的改变
                    8'h1C : begin //左
                    //4'b1000: begin
                        if(ballY>=1) begin
                            if(Map[ballX][ballY-
1]==0||Map[ballX][ballY-1]==2) begin
                                ballY_now<=ballY-1; ballX_now<=ballX;
                            end
                            else begin
                                //click<=1;
                                ballX_now=ballX; ballY_now=ballY;
                            end
                        end
                        else begin
                            ballX_now=ballX; ballY_now=ballY;
                        end
                        if(ballX==y_height && ballY==x_length) begin
                            ballX_now<=0; ballY_now<=0;
                        end
                    end
                    8'h23 : begin //右
                    //4'b0100: begin
                        if(ballY<=x_length-1) begin

if(Map[ballX][ballY+1]==0||Map[ballX][ballY+1]==2)begin
                            ballY_now<=ballY+1;  ballX_now<=ballX;
                            end
                            else begin
                                ballX_now=ballX; ballY_now=ballY;
                            end
                        end
                        else begin
                            ballX_now=ballX; ballY_now=ballY;
```

```
                        end
                        if(ballX==y_height && ballY==x_length) begin
                            ballX_now<=0; ballY_now<=0;
                        end
                    end
                    8'h1B : begin //下
                    //4'b0010: begin
                        if(ballX<=y_height-1) begin
if(Map[ballX+1][ballY]==0||Map[ballX+1][ballY]==2)begin
                            ballX_now=ballX+1;  ballY_now=ballY;  click=0;
                            end
                            else begin
                                ballX_now=ballX; ballY_now=ballY;
                            end
                        end
                        else begin
                            ballX_now=ballX; ballY_now=ballY;
                        end
                        if(ballX==y_height && ballY==x_length) begin
                            ballX_now<=0; ballY_now<=0;
                        end
                    end
                    8'h1D : begin //上
                        if(ballX>=1) begin
                            if(Map[ballX-1][ballY]==0||Map[ballX-
1][ballY]==2)begin
                            ballX_now=ballX-1; ballY_now=ballY;
                            end
                            else begin
                                ballX_now=ballX; ballY_now=ballY;
                            end
                        end
                        else begin
                            ballX_now=ballX; ballY_now=ballY;
                        end
                        if(ballX==y_height && ballY==x_length) begin
                            ballX_now=0; ballY_now=0;
                        end
                    end
```

```
                    default: begin
                        ballX_now=ballX; ballY_now=ballY;
                    end
                endcase
        end
    end
    //时钟分频
    //2 分频
    always @(posedge CLK or posedge aclr_i) begin
        if(aclr_i) begin
            SREG <= 1'b0;
        end
        else begin
            SREG <= ~SREG;
        end
    end
    //4 分频
    always @(posedge CLK or posedge aclr_i) begin
        if(aclr_i) begin
            CLKOUT <= 1'b0;
        end
        else if(SREG) begin
            CLKOUT <= ~CLKOUT;
        end
    end
endmodule
```

4.4 MP3 编码模块

MP3 编码模块主要负责接收 MP3 硬件数据请求（DREQ），并传送数据（XRSET / XCS / XDCS / SI / SCLK），分为七个状态，可以互相转换：

状态	状态码	作用
CMD_START	0	开始写指令
WRITE_CMD	1	将一条指令全部写入
DATA_START	2	开始写数据
WRITE_DATA	3	将一条数据全部写入
DELAY	4	延时
VOL_CMD_START	5	写音量修改指令
SEND_VOL_CMD	6	CMD 指令发送

IP 核中导入了四种提示音的 COE 文件：

```
//选择提示音
blk_mem_gen_0 your_instance_name0(.clka(CLK),.addra(addr),.douta(D_do));
blk_mem_gen_1 your_instance_name1(.clka(CLK),.addra(addr),.douta(D_re));
blk_mem_gen_2 your_instance_name2(.clka(CLK),.addra(addr),.douta(D_mi));
blk_mem_gen_3 your_instance_name3(.clka(CLK),.addra(addr),.douta(D_fa));
```

对应了获取键码的 W/A/S/D，即使用键盘移动小球时，上下左右分别发出不同音效。

接口定义：

```
module MP3(
    input CLK,          //系统时钟
    input kbstrobe_i,   //去抖信号
    input DREQ,         //数据请求
    output reg XRSET,   //硬件复位
    output reg XCS,     //低电平有效片选输出
    output reg XDCS,    //数据片字节同步
    output reg SI,      //串行数据输入
    output reg SCLK,    //SPI 时钟
    input init,         //初始化
    input [3:0] hex1,   //16 进制键码高位
     input [3:0] hex0,   //16 进制键码低位
     input [15:0] adjusted_vol,  //实时音量
     input keyup,
     output reg [3:0] tune=0      //当前按下指令对应提示音种类
);
```

模块建模：

```
module MP3(
    input CLK,          //系统时钟
    input kbstrobe_i,   //去抖信号
    input DREQ,         //数据请求
    output reg XRSET,   //硬件复位
    output reg XCS,
    //低电平有效片选输出
    output reg XDCS,
    //数据片字节同步
    output reg SI,  //串行数据输入
    output reg SCLK,    //SPI 时钟
    input init,         //初始化
```

```
    input [3:0] hex1,
    //16 进制键码高位
    input [3:0] hex0,
    //16 进制键码低位
    input [15:0] adjusted_vol,
    //实时音量
    input keyup,
    output reg [3:0] tune=0
    //当前按下指令对应提示音种类
);
    parameter  CMD_START=0;
    //开始写指令
    parameter  WRITE_CMD=1;
    //将一条指令全部写入
    parameter  DATA_START=2;
    //开始写数据
    parameter  WRITE_DATA=3;
    //将一条数据全部写入
    parameter  DELAY=4;
    //延时
    parameter VOL_CMD_START=5;
    //音量相关
    parameter SEND_VOL_CMD=6;
    //CMD 发送

    reg [31:0] volcmd;
    reg [20:0]addr;
    wire CLK_1M; //分频 1MHz
    Divider #(.N(100)) CLKDIV1(CLK,1,clk_1M);

    reg  [15:0] Data;
    wire [15:0] D_do;
    wire [15:0] D_re;
    wire [15:0] D_mi;
    wire [15:0] D_fa;

    reg [3:0] pretune=0;
    reg [15:0] _Data;
```

```
    //选择提示音
    blk_mem_gen_0
your_instance_name0(.clka(CLK),.addra(addr),.douta(D_do));
    blk_mem_gen_1
your_instance_name1(.clka(CLK),.addra(addr),.douta(D_re));
    blk_mem_gen_2
your_instance_name2(.clka(CLK),.addra(addr),.douta(D_mi));
    blk_mem_gen_3
your_instance_name3(.clka(CLK),.addra(addr),.douta(D_fa));

    integer tune_delay=0;       //延时 50000 单位
    always @(posedge CLK_1M)
    begin
       if(tune_delay==0)
       begin
            if(keyup&&kbstrobe_i)
            begin
               tune_delay<=50000;
               case({hex1,hex0})
//根据不同的键码发出不同声音
               8'h1C:
               begin
                   tune<=4'b0001;
// A->左移
               end
               8'h1B:
               begin
                    tune<=4'b0010;  // S->下移
               end
               8'h23:
               begin
                    tune<=4'b0011;  // D->右移
               end
               8'h1D:
               begin
                    tune<=4'b0100;  // W->上移
               end
               default:
               begin
```

```
                    tune<=0;
                end
                endcase
              end
        end
        else
        begin
             tune_delay<=tune_delay-1;
//延时读取指令
        end
        case(tune)
        4'b0001:
        begin
            Data<=D_do;
        end
        4'b0010:
        begin
            Data<=D_re;
        end
        4'b0011:
        begin
            Data<=D_mi;
        end
        4'b0100:
        begin
            Data<=D_fa;
        end
        default:
        begin
            Data<=D_do;
        end
        endcase
    end

    reg [63:0] cmd={32'h02000804,32'h020B0000};
//00 是控制模式 0B 是音量 08 本地模式 04 软件复位（每首歌之后软件复位）
    //开始写指令
    integer status=CMD_START;
    integer cnt=0;      //位计数
```

```
    integer cmd_cnt=0;  //命令计数

    always @(posedge CLK_1M)
    begin
        pretune<=tune;
//存储 tune，便于同步接收下一个 tune
if(~init||pretune!=tune||!keyup)
//以下情况（复位或可能出现错误）回到初始状态
        begin
            XCS<=1;
            XDCS<=1;
            XRSET<=0;
            cmd_cnt<=0;
            status<=DELAY;
// 刚开机时先 delay,等待 DREQ
            SCLK<=0;
            cnt<=0;
            addr<=0;
        end
        else if((tune<4'b0101&&addr<10000))
        begin
            case(status)
            CMD_START:
// 等待允许输入
           begin
                SCLK<=0;
//SPI 时钟下降沿输入，上升沿读取，创造下降沿
                if(cmd_cnt>=2)
// 把前 2 组预设命令（mode 音量 ）输入完毕后 开始输入音调
                  status<=DATA_START;
                else if(DREQ)
// DREQ 有效时 允许输入 si 可以接受 32（bit）信号
                begin
                    XCS<=0;//XCS 拉低表示输入指令
                    status<=WRITE_CMD;
// 开始输入指令
                    SI<=cmd[63];
  cmd<={cmd[62:0],cmd[63]};
//移位传数据
```

```
                    cnt<=1;
                end
            end
            WRITE_CMD://写入指令
            begin
                if(DREQ)
                begin
                    if(SCLK)
                    begin
                        if(cnt>=32)     //位计数
                        begin
                            XCS<=1;  // 取消复位
                            cnt<=0;
cmd_cnt<=cmd_cnt+1;
//共发送两条 cmd 指令
       status<=CMD_START;
// 跳转到命令执行
                        end
                        else
                        begin
SCLK<=0;
SI<=cmd[63];
// 发送三十二位写指令（写指令 0200（择 MODE 寄存器） 0804（MODE 有十六位 这里只关心第 11 和 2 位 进行软复位）
cmd<={cmd[62:0],cmd[63]};
//循环传送
cnt<=cnt+1;
                        end
                    end
                    SCLK<=~SCLK;
                end
            end
            DATA_START://写入数据
            begin
if(adjusted_vol[15:0]!=cmd[47:32])  // cmd[47:32] 里存储的是当前音量 初始值为 0000
                begin//音量变了
                    cnt<=0;
volcmd<={16'h020B,adjusted_vol}; // 调节音量命令 0B 寄存器负责音量控剜  该
```

寄存器存储内容表示音量大小

```
status<=VOL_CMD_START;            // 转到音量调节
                end
                else if(DREQ)
    // 等待允许输入 之后每次输入 32 使 等 DREQ 下次变高接着输入 vs1003B 会自动接收并播放
                begin
//不需要调节音量，直接转到音频数据传送
                    XDCS<=0;
                    SCLK<=0;
                    SI<=Data[15];
_Data<={Data[14:0],Data[15]};
                    cnt<=1;
status<=WRITE_DATA;  // 歌曲信号
                end
cmd[15:0]<=adjusted_vol;
// 更新 cmd 中存储的音量
            end
            WRITE_DATA:
            begin
                if(SCLK)
                begin
                    if(cnt>=16)
                    begin
                        XDCS<=1;
addr<=addr+1;
// 读完十六位后地址后移 1 位
status<=DATA_START;
                    end
                    else
                    begin
// 循环输入 输入十六位
                        SCLK<=0;
                        cnt<=cnt+1;
_Data<={_Data[14:0],_Data[15]};
SI<=_Data[15];
                    end
                end
                SCLK<=~SCLK;
```

```
            end
            DELAY:
            begin
                if(cnt<50000)
// 等待 100 个时钟周束
                    cnt<=cnt+1;
                else
                begin
                    cnt<=0;
status<=CMD_START;
                    XRSET<=1;
                end
            end
            VOL_CMD_START:
            begin
                if(DREQ)
                begin
// 等待 DREQ 信号
                    XCS<=0;
status<=SEND_VOL_CMD;
// 输入音量控制命令
                    SI<=volcmd[31];
volcmd<={volcmd[30:0],volcmd[31]};
                    cnt<=1;
                end
            end
            SEND_VOL_CMD:
            begin
                if(DREQ)
                begin
                     if(SCLK)
                     begin
                        if(cnt<32)  //循环传值
                        begin
SI<=volcmd[31];
volcmd<={volcmd[30:0],volcmd[31]};
cnt<=cnt+1;
                        end
                        else
```

```
                        begin
                            XCS<=1; // 结束输入
                            cnt<=0;
status<=DATA_START;
                        end
                    end
                    SCLK<=~SCLK;
                end
            end
        default:;
        endcase
    end
end
endmodule
```

4.5 MP3 调音模块

MP3 调音原理即为获取键盘键码，若为调音按键，则改变音量大小 vol 和音量等级 vol_class，并将音量等级（0~9）直观地显示在 N4 开发板的七段数码管上。

需要注意的是音量值越小，实际声音越大，音量等级也越大。本系统中音量等级和音量实际大小对应如下表：

音量值 vol	**音量等级 vol_class**
16'he577	0
16'hcbf8	1
16'hb279	2
16'h98fa	3
16'h7f7b	4
16'h65fc	5
16'h4c7d	6
16'h32fe	7
16'h197f	8
16'h0000	9

通过按键（小键盘）KP_2 和 KP_8 分别控制音量的减小和增大。

接口定义：

```
module VolAdjust(
    input clk,
    input [3:0] hex1,
    input [3:0] hex0,
    input keyup,
    output reg [3:0] vol_class=9,  //音量等级 初始最大音量
```

```
    output reg [15:0]vol=16'h0000   //音量数额
    );
```

模块建模：

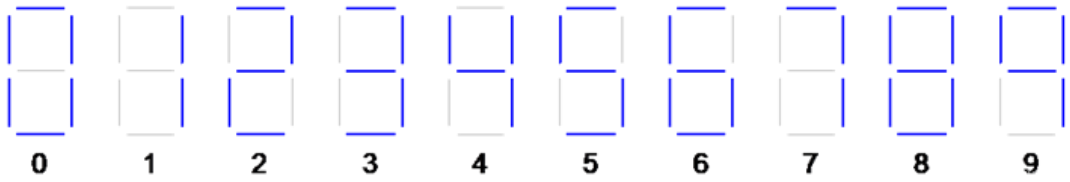
```
module VolAdjust(
    input clk,
    input [3:0] hex1,
    input [3:0] hex0,
    input keyup,
    output reg [3:0] vol_class=9,  //音量等级 初始最大音量
    output reg [15:0]vol=16'h0000   //音量数额
    );
    wire [15:0]adjusted_vol;        //调整后的音量
    wire CLK_1M; //分频 1MHz
    Divider #(.N(100)) CLKDIV1(CLK,1,clk_1M);
    assign adjusted_vol=vol;        //实时存储原音量大小
    integer clk_cnt=0;              //时钟周期计数器，用于设置延时
    always @(posedge clk_1M)
    begin
        if(clk_cnt==200000) begin
            clk_cnt<=0;
            if(!keyup) begin
                case({hex1,hex0})
                8'b01110101:
//小键盘 KP_8
                begin
                    vol<=(vol==16'h0000)?16'h0000:(vol-16'h197f);
// 音量放大（vol 值变小）
                end
                8'b01110010:
//小键盘 KP_2
                begin
vol<=(vol==16'hfef6)?16'hfefe:(vol+16'h197f);
//音量减小（vol 值变大）
                end
                endcase
            end
        end
        else
```

```
        clk_cnt<=clk_cnt+1;
end
//更新 vol_class 音量等级
always @(posedge clk_1M)
begin
    case(vol)
    16'he577:
    begin
        vol_class<=0;
    end
    16'hcbf8:
    begin
        vol_class<=1;
    end
    16'hb279:
    begin
        vol_class<=2;
    end
    16'h98fa:
    begin
        vol_class<=3;
    end
    16'h7f7b:
    begin
        vol_class<=4;
    end
    16'h65fc:
    begin
        vol_class<=5;
    end
    16'h4c7d:
    begin
        vol_class<=6;
    end
    16'h32fe:
    begin
        vol_class<=7;
    end
    16'h197f:
```

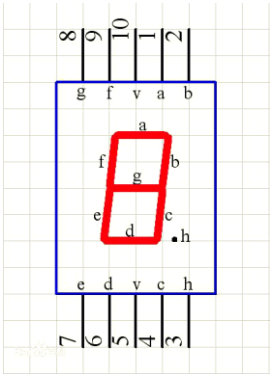
```
        begin
            vol_class<=8;
        end
        16'h0000:
        begin
            vol_class<=9;
        end
        default:
        begin
            vol_class<=0;
        end
        endcase
    end
endmodule
```

4.6 七段数码管模块

使用七段数码管数字化显示音量等级（MP3 模块中的 vol_class），共 0~9 十个数字，数字大小和音量大小成正比（和音量值 vol 成反比）。



其中数字和模块中 oData 输出值显示遵循以下表格规范：



显示数字	oData
0	1000000
1	1111001
2	0100100
3	0110000
4	0011001
5	0010010
6	0000010
7	1111000
8	0000000
9	0010000

接口定义：

```
module display7(iData,init,oData);
    input [3:0] iData;
    input init;
    output [6:0] oData;
```

模块建模：

```
module display7(iData,init,oData);
    input [3:0] iData;
    input init;
    output [6:0] oData;
    reg [6:0] oData;
    initial
    oData=7'b1111111;
    always@(*)
        case({iData})
            4'b0000: oData=7'b1000000;
            4'b0001: oData=7'b1111001;
            4'b0010: oData=7'b0100100;
            4'b0011: oData=7'b0110000;
            4'b0100: oData=7'b0011001;
            4'b0101: oData=7'b0010010;
            4'b0110: oData=7'b0000010;
            4'b0111: oData=7'b1111000;
            4'b1000: oData=7'b0000000;
            4'b1001: oData=7'b0010000;
            default: oData=7'b0000000;
        endcase
endmodule
```

4.7 分频模块

由于 MP3 模块等多个模块需要 1MHz 频率的时钟信号，故需要将系统时钟分频 100 倍使用该分频模块，传入分频值 N。

接口定义：

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

模块建模：

```
module Divider(
```

```
    input I_CLK,
    input rst,
    output reg O_CLK
    );
    parameter N=100000000;
    integer count=0;
    always @(negedge rst or posedge I_CLK )
        begin
            if(!rst)
                O_CLK=0;
            else
            begin
                if(count==N)
                    begin
                        O_CLK=~O_CLK;
                        count=0;
                    end
                else
                    count=count+1;
            end
        end
    endmodule
```

调用方法：

```
wire CLK_1M;       //分频 1MHz
Divider #(.N(100)) CLKDIV1(CLK,1,clk_1M);
```

4.8 信号去抖模块

时序逻辑中易出现因抖动造成的尖峰信号脉冲，对游戏控制造成极大影响，故引入去抖模块将 keyup 信号去抖处理。

接口定义：

```
module SwitchDB(
    input CLK,//系统时钟
    input SW,//是否读取到键的信息，1 为读取到
    input ACLR_L,//控制信号
    output reg SWDB//去抖信号
    );
```

模块建模：

```
module SwitchDB(
    input CLK,//系统时钟
    input SW,//是否读取到键的信息，1 为读取到
    input ACLR_L,//控制信号
    output reg SWDB//去抖信号
    );
wire aclr_i;
reg CLKOUT;
reg [1:0] Q_CURR;//状态机的状态
parameter [1:0] SW_OFF = 2'b00; //状态机定义
parameter [1:0] SW_EDGE = 2'b01;
parameter [1:0] SW_VERF = 2'b10;
parameter [1:0] SW_HOLD = 2'b11;
assign aclr_i = ~ACLR_L;
//状态转移
always @(posedge CLKOUT or posedge aclr_i) begin
    SWDB <= 1'b0;   //默认为 0
    if(aclr_i) begin
        Q_CURR <= SW_OFF;
    end
    else begin
        case(Q_CURR)
            SW_OFF: if(SW) begin
                        Q_CURR <= SW_EDGE;
                    end
                    else begin
                        Q_CURR <= SW_OFF;
                    end

            SW_EDGE: if(SW) begin
                        Q_CURR <= SW_VERF;
                        SWDB <= 1'b1;
//去抖信号仅在当前状态为 SW_VERF 时为高
                     end
                     else begin
                        Q_CURR <= SW_OFF;
                     end
```

```
            SW_VERF: Q_CURR <= SW_HOLD;

            SW_HOLD: if(SW) begin
                        Q_CURR <= SW_HOLD;
                     end
                     else begin
                        Q_CURR <= SW_OFF;
                     end
        endcase
    end
end
reg SREG;
//时钟分频
//2 分频
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        SREG <= 1'b0;
    end
    else begin
        SREG <= ~SREG;
    end
end
//4 分频
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        CLKOUT <= 1'b0;
    end
    else if(SREG) begin
        CLKOUT <= ~CLKOUT;
    end
end
endmodule
```

4.9 顶层模块

系统顶层模块用于处理各种子系统的调用及参数的传递，以及整个系统的输入输出参数设计。

接口定义：

```
module MazeCity(
    input CLK,
    input RST,
    //VGA 显示器部分
    output HSYNC,           //帧同步信号
    output VSYNC,           //行同步信号
    output [3:0] RED,
    output [3:0] GREEN,
    output [3:0] BLUE,
    //键盘
    input usbCLK,               //F4
    input usbDATA,              //usb 数据 B2
    output click,               //检测信号
    //mp3
    input DREQ,//数据请求
    output wire XRSET,          //硬件复位
    output wire XCS,            //低电平有效片选输藝
    output wire XDCS,           //数据片字节同步
    output wire SI,             //串行数据输入
    output wire SCLK,           //SPI 时钟
    //七段数码管
    output wire [6:0] oData
    );
```

模块建模：

```
module MazeCity(
    input CLK,
    input RST,
    //VGA 显示器部分
    output HSYNC,           //帧同步信号
    output VSYNC,           //行同步信号
    output [3:0] RED,
    output [3:0] GREEN,
    output [3:0] BLUE,
    //键盘
    input usbCLK,               //F4
    input usbDATA,              //usb 数据 B2
    output click,               //检测信号
```

```
    //mp3
    input DREQ,//数据请求
    output wire XRSET,          //硬件复位
    output wire XCS,            //低电平有效片选输藝
    output wire XDCS,           //数据片字节同步
    output wire SI,             //串行数据输入
    output wire SCLK,           //SPI 时钟
    //七段数码管
    output wire [6:0] oData
    );
    wire keyup;
    wire [11:0] CSEL;
    wire [9:0] HCOORD,VCOORD;
    wire [3:0] hex1,hex0;
    wire [5:0] ballX,ballY;
    wire [15:0] adjusted_vol;
    wire [3:0] vol_class;
    wire [3:0] tune;
    wire kbstrobe_i;        //去抖信号
    Keyboard
KB(.CLK(usbCLK), .SDATA(usbDATA), .ARST_L(RST), .HEX1(hex1), .HEX0(hex0
), .KEYUP(keyup));
    SwitchDB switch_DB(CLK,keyup,RST,kbstrobe_i);
    VGA
vga_Display(.CLK(CLK),.CSEL(CSEL),.ARST_L(RST),.HSYNC(HSYNC),.VSYNC(VSY
NC),.RED(RED),.GREEN(GREEN),.BLUE(BLUE),.HCOORD(HCOORD),.VCOORD(VCOORD)
);
    VGA_controller
vga_control(CLK,kbstrobe_i,{hex1,hex0},HCOORD,VCOORD,RST,ballX,ballY,ba
llX,ballY,CSEL,click);
    VolAdjust
vol_adj(CLK,hex1,hex0,keyup,vol_class,adjusted_vol,click,kbstrobe_i);
    MP3
mp3(CLK,kbstrobe_i,DREQ,XRSET,XCS,XDCS,SI,SCLK,RST,hex1,hex0,adjusted_v
ol,keyup,tune,click);
    display7 Display(vol_class,oData);

endmodule
```

五、测试模块建模

5.1 键盘编码模块 TB

```
module KBDecoderTestBench;
    reg CLK, ARST_L, SDATA;
    wire KEYUP, rollover;
    wire [3:0] HEX0, HEX1;
    reg [21:0] in_seq;
    wire [4:0] SREG;
Keyboard
DUT(.CLK(CLK), .ARST_L(ARST_L), .SDATA(SDATA), .KEYUP(KEYUP), .HEX0(HEX
0), .HEX1(HEX1), .SREG(SREG), .rollover_i(rollover));
initial in_seq <= 22'b1100000100111111100001;
initial CLK <= 1'b0;
always  #`clktemp CLK <= ~CLK;
initial
begin
    ARST_L <= 1'b0;
    #25 ARST_L <= 1'b1;
end

always @(posedge CLK)
    if(ARST_L == 1'b0)
        SDATA <= #1 1'b0;
    else
        SDATA <= #1 in_seq[0];
always @(posedge CLK)
    if(ARST_L == 1'b0)
        in_seq <= #1 in_seq;
    else
        in_seq <= #1 {1'b0, in_seq[21:1]};
endmodule
```

5.2 VGA 控制模块 TB

```
module VGAController_TB;
    reg CLK, ARST_L;
    wire rollover_i, aclr_i;
```

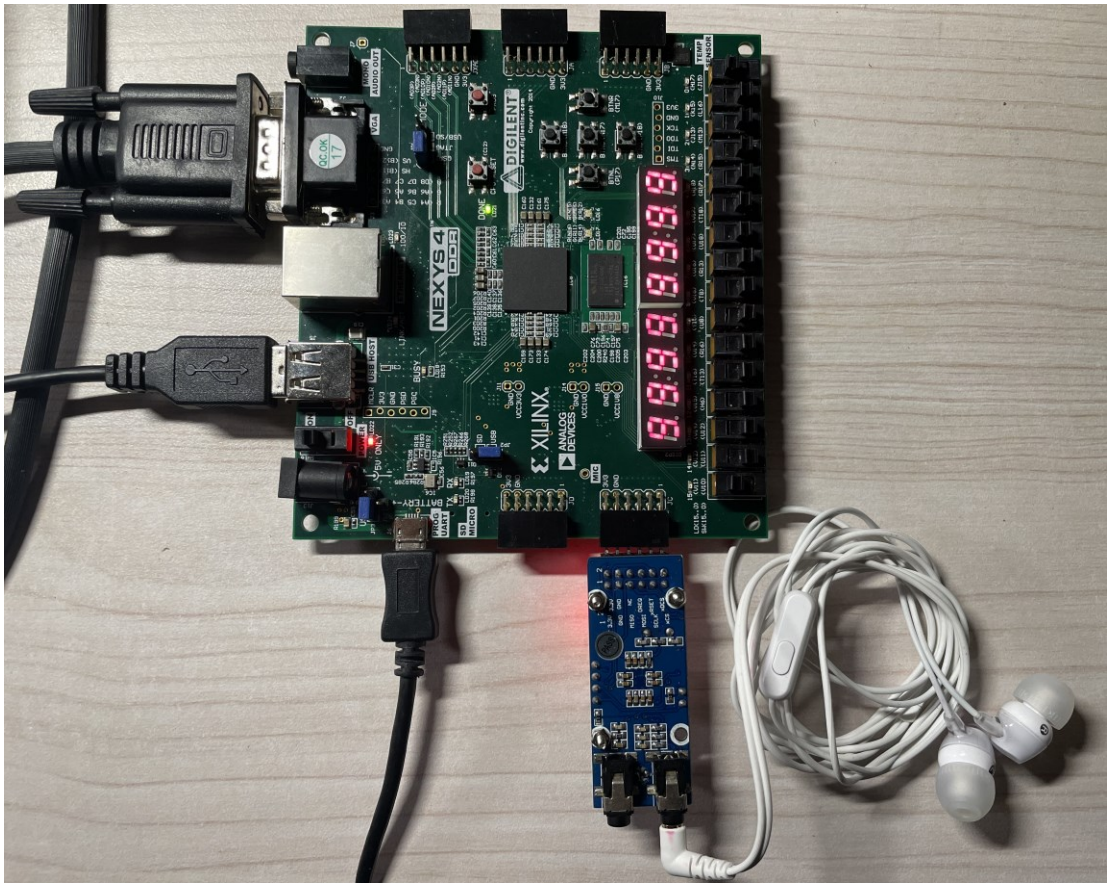
```
    wire Hrollover_i, Vrollover_i;
    wire [17:0] SREG;
    wire [11:0] CSEL;
    reg [9:0] HCOORD, VCOORD;
    wire [9:0] XPos, YPos;
    wire [5:0] XSpeed, YSpeed;
VGAController
DUT(.CLK(CLK), .ARST_L(ARST_L), .SREG(SREG), .rollover_i(rollover_i), .
HCOORD(HCOORD), .VCOORD(VCOORD), .CSEL(CSEL), .XPos(XPos), .YPos(YPos),
 .XSpeed(XSpeed), .YSpeed(YSpeed));
assign aclr_i = ~ARST_L;
initial CLK <= 1'b0;
always  #`clktemp CLK <= ~CLK;
initial
begin
    ARST_L <= 1'b0;
    #25 ARST_L <= 1'b1;
end
assign Hrollover_i = (HCOORD[9] & HCOORD[8] & HCOORD[5]) ? 1'b1 : 1'b0;
assign Vrollover_i = (VCOORD[9] & VCOORD[3] & VCOORD[2] & VCOORD[0]) ?
1'b1 : 1'b0;
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        HCOORD <= 10'b0000000000;
    end
    else if(Hrollover_i)
        HCOORD <= 10'b0000000000;
    else
        HCOORD <= HCOORD + 1;
end
always @(posedge CLK or posedge aclr_i) begin
    if(aclr_i) begin
        VCOORD <= 10'b0000000000;
    end
    else if(Vrollover_i)
        VCOORD <= 10'b0000000000;
    else if(Hrollover_i)
        VCOORD <= VCOORD + 1;
end
```

```
endmodule
```

六、实验结果

下板试验结果如下

N4 开发板连线：



左侧从上至下分别是 VGA 接口和键盘 USB 接口。VGA 显示正常（见下图）；键盘可控制且反应迅速；耳机可正常播放不同音效（且音量可受键盘控制调节）；七段数码管可正常显示音量等级。

VGA 显示屏：

（实际色彩显示更为鲜亮）

左上角为迷宫起点，右下角为迷宫终点，可通过键盘控制小球移动到达终点后，重新开始游戏。

七、总结

通过本次大作业，让我了解到了包括 PS2 协议、USB-hid 和 SPI 协议等知识，并通过 Verilog 语言尝试实现以上协议，并自行实践了串行外接设备端口，最终将 VS100SB-MP3 模块、PS2 键盘模块和 VGA 显示器模块与 N4 开发板连接，搭建了完备的、可拓展的、面向幼儿群体的迷宫益智游戏。

- “**完备性**”体现在每个部件都得到充分的利用，既可以通过键盘控制小球位置、也可以改变音量大小，N4 开发板即是所有模块开发和外设连接的载体、也通过自身的七段数码管数字化显示音量，此外 MP3 和 VGA 也得到了完善利用；
- “**可拓展性**”体现在部分可以在模块内存储的数据（如小球位置等）传到模块外，便于后期增添模块使用；此外，迷宫地图 Map 数组使用寄存器 reg 类型存储，在 initial 块中赋初值，而不是使用常量 parameter 定义，便于后期添加用户修改地图或随机地图的新模块。
- “**面向幼儿群体**”体现在由于使用 40*40px 色块绘制，颜色鲜亮明显，适合幼儿在保护视力的距离下看清屏幕内容。墙体区别于单纯色块，使用 8px 宽斜条纹绘制，可让幼儿群体拥有一双发现美的眼睛，观察生活中的像素级美学。小球上绘制笑脸，有助于在幼儿时期培养乐观开朗的人生态度，微笑面对未来生活中的一切。关卡设计简单，但设计多处陷阱，需要不断尝试，寓意接下来的人生道路中难免会有壁障和歧

途，但是只要有一颗敢于尝试的心，一定能到达成功的彼岸。

八、心得体会及建议

8.1 课程心得体会

经过一学期的数字逻辑课程学习，我对数字电路和计算机组成原理等方面的知识有了一定了解，为后续计算机组成原理课程的学习打下基础。

首先学习了布尔代数，了解了布尔代数的基本概念，包括逻辑运算、真值表、布尔函数、布尔恒等式、卡诺图等，同时掌握如何将自然语言的问题转换为布尔代数表达式。

接着学习了组合逻辑电路，即组合逻辑电路的设计方法和分析技术，包括逻辑门、编码器、译码器、多路选择器、加法器、比较器等电路的设计和实现，掌握了使用逻辑门和布尔代数设计和实现简单的数字电路。

其次是时序逻辑电路，学习了时序逻辑电路的设计和分析方法，包括触发器、计数器、寄存器、状态机等电路的设计和实现，掌握了时序逻辑电路的时序行为，以及如何使用状态图、状态转移表等工具来描述和分析时序电路。

然后了解了存储器和处理器，具体来说是存储器和处理器的基本结构和工作原理，包括内存、缓存、硬盘、CPU 等组成部分的功能和实现，掌握了存储器和处理器的层次结构、指令集架构、流水线等基本概念。

最后是数字系统设计，即数字系统设计的基本方法和流程，包括需求分析、规格说明、系统设计、仿真验证等环节，掌握了数字系统设计的基本思想和方法。

此外，还学习了硬件描述语言 Verilog 及其设计方法，在本次大作业中得到充分的利用。

除了以上知识点，由于数字逻辑是一门涉及硬件的课程，对于大一只有高程基础的我们是一个挑战，但是通过充分理解和运用知识点，以及数字逻辑和计算机科学导论两门课程的结合，让我对计算机的组成架构、运行方式等有了一个更充分的了解——计科导课程描述了计算机的整体架构、运行方式和各个部件的协作方式，如了解了 CPU 的组成；数字逻辑则从更微观的角度（通常以 bit 位为单位）进行具体部件的分析，如 CPU 中具体的寄存器等。因此从更微观的角度了解计算机，也能引发日常生活中使用计算机时，对其特性的思考。

8.2 课程心得体会

1. 希望老师平时更注重整体思维的培养，涉及到部件或理论方法时课程内容较为分散，难以形成整体性。
2. 希望实验课前后内容循序渐进，而不是单纯部件的实验，如下一次实验可以和上一次实验的部件相结合，组合为具有创新功能的系统。
3. 希望平时注重培养发散性思维，将计算机中可以察觉的现象和课程理论知识结合起来，可以引起平时对计算机组成和运行方式的思考。

8.3 数字芯片设计的认识与体会

近日，美国对展开中国新一轮芯片制裁，主要涉及禁止向中国出口高端芯片和关键技术，以及对中国公司和个人实施制裁。这些措施对中国芯片制造业的发展将造成严重的影响。

我认为美国对中国新一轮芯片制裁的背景是出于美国政府对中国数字芯片行业快速崛起和发展的担忧。这些制裁措施旨在限制中国芯片制造业的发展，以保持美国在国际上的竞争地位。

对于中国数字芯片行业而言，这种制裁显然是一个不利因素，将对中国芯片制造业的发展造成一定程度的阻碍。然而，这也是一个机遇，迫使中国数字芯片行业加强自主创新，提高核心技术的自主掌握，以减轻外部压力对其发展的影响。

在应对制裁的过程中，中国数字芯片行业需要加强与国际上其他先进芯片制造国家的合作和交流，借鉴和吸收国际上最新的技术和理念，以推动数字芯片行业的快速发展。此外，行业内也需要加强人才培养，吸引和培养更多的高级芯片设计师和工程师，以提高数字芯片行业的整体水平。

总的来说，中国数字芯片行业需要以此次制裁为契机，把握当前机遇，加强自主创新能力，与国际上其他先进芯片制造国家合作，加强人才培养，以推动数字芯片行业的快速发展。