同济大学计算机系

数字逻辑课程综合实验报告

目录

一、实验内容

1. 项目内容概括

基于FPGA 、蓝牙模块、VGA以及声音传感器的Flappy Bird小游戏

2. 界面样式

如图所示，界面包括游戏开始图片、游戏中方块、挡板、游戏结束图片

3. 操作说明

刚开始时为游戏开始画面。通过蓝牙模块，用外部设备（电脑、手机等）发送非零信号进入游戏画面，并且七段数码管开始计时。同时，蓝牙接收的数据将影响挡板移动速度，实现游戏的不同难度。向声音传感器发出声音，使得方块向上移动。若再次发出声音，则方块向下移动。若方块触碰到挡板，游戏结束，跳转到游戏结束画面。同时，七段数码管停止计时。若想重新开始游戏，则向蓝牙发送零信号，返回游戏开始画面

4. 规则说明

打开游戏开关后，七段数码管计时开始。游戏结束后七段数码管停止计时。可通过蓝牙调整游戏难度。

5. 器件简介

5.1 Bluetooth Slave UART Board——蓝牙模块，其协议为UART协议。可进行无线蓝牙数据传输。

5.2 Sound Sensor——声音传感器，可根据外部声音产生信号。

5.3 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。

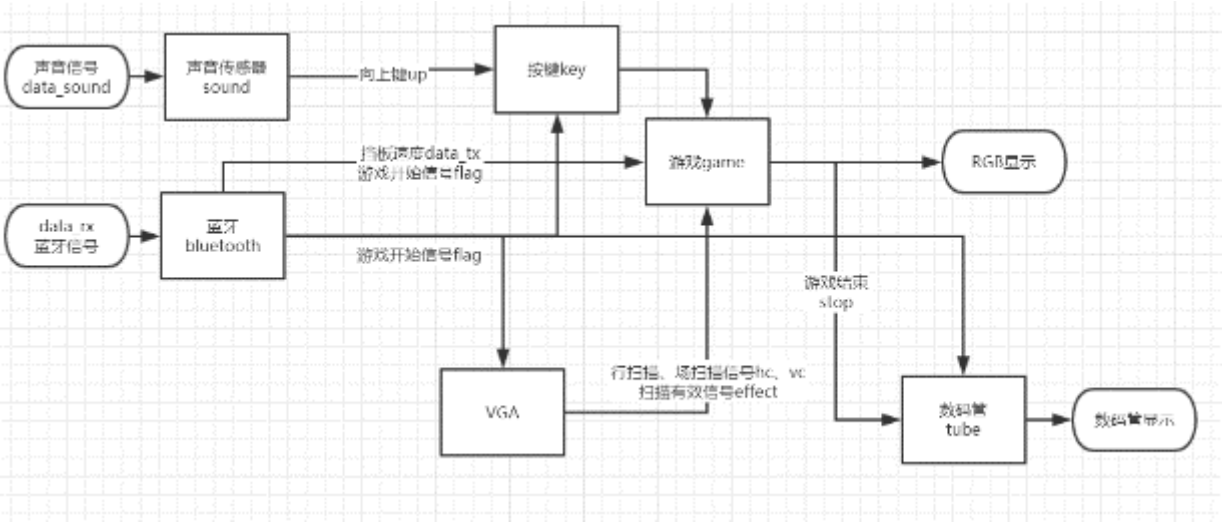
5.4 Nexys 4 DDR Artix-7——由 Xilinx 公司开发出的一款现场可编程门阵列（FPGA） 开发板

二、数字系统总框图

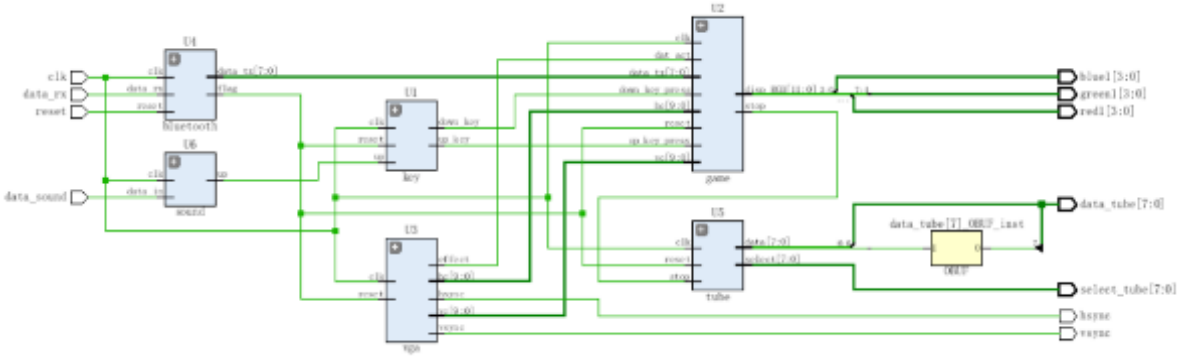
要实现的功能如下：

1. 实现VGA显示
2. 实现声音传感器的通信和防抖
3. 实现蓝牙模块的通信
4. 实现基于七段数码管的数字钟计时
5. 实现游戏的按键
6. 实现游戏本身
7. 实现基于ROM的图片显示
8. 实现由游戏开始到游戏进行再到图片结束再到游戏开始的转换
9. 实现蓝牙对游戏和七段数码管的控制
10. 实现声音传感器对按键的控制

综上所述，最终总框图设计如下：

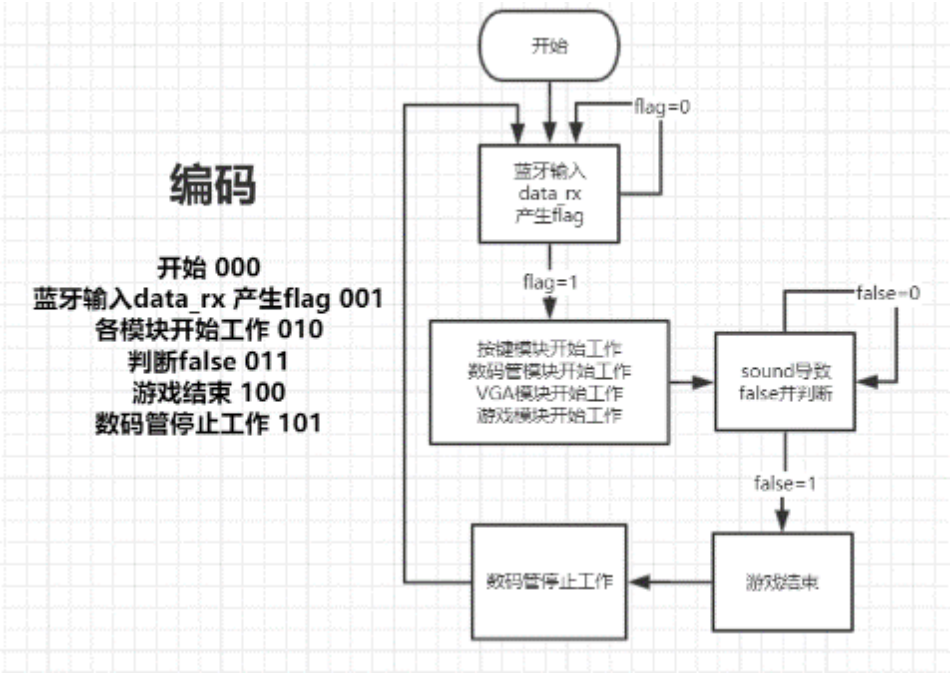


RTL图如下：



三、系统控制器设计

1.ASM 流程图



2.状态转移真值表

PS	NS	转换条件
000	001	/
001	001	0X
001	010	1X
010	011	/
011	011	10
011	100	11
100	101	/
101	001	/

3.次态激励函数表达式

令转移条件flag为X，false为Y，编码首位为A，次位为B，末位为C，得：

A^n+1=A~B~C+~ABCXY

B^n+1=~A(CX~Y+~BC)

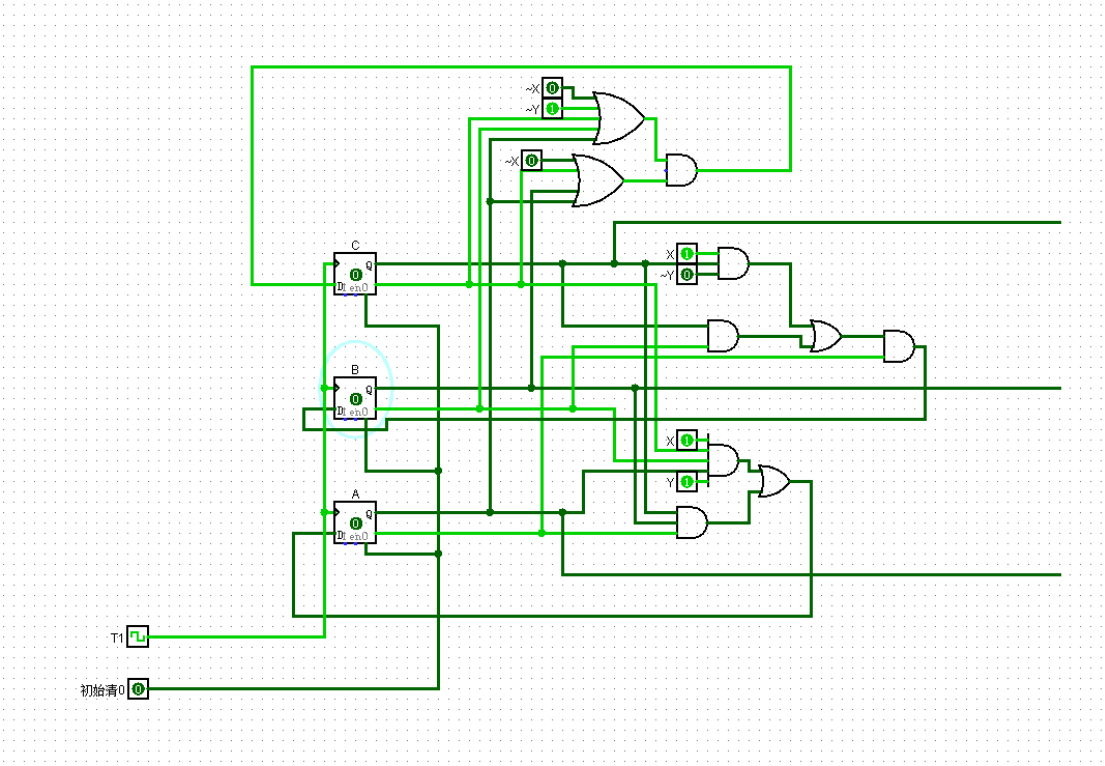
C^n+1=(A+B+~C+~X)(A+~B+~C+~X+~Y)

4.控制命令逻辑表达式

令((rand<move_y + border)&&(move_y < rand0+long)&&(push < move_x+border) && (move_x < push + ban)) 为 M，sound为N

Y^n+1=M*N

5.系统控制器逻辑方案图



四、子系统模块建模

1.top顶层模块

功能：连接各个子模块，列出了所有的与外部交流的数据。

接口信号定义：
```
input clk;//时钟
input reset;//蓝牙清零（无用）
input data_rx;//蓝牙输入
input data_sound;//声音输入
output hsync; //VGA 行同步信号
output vsync; //VGA 场同步信号
output [3:0]red1;//输出红色的4位数据
output [3:0]green1; //输出绿色的4位数据
output [3:0]blue1; //输出蓝色的4位数据
output [7:0] data_tube;//数码管段选数据
output [7:0] select_tube;//数码管位选
```

代码：

```
module
```

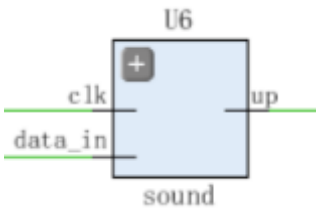
```
top(clk,reset,data_rx,data_sound,hsync,vsync,red1,green1,blue1,data_tube,select_tube);
input clk;//时钟
input reset;//蓝牙清零（无用）
input data_rx;//蓝牙输入
input data_sound;//声音输入

output hsync; //VGA 行同步信号
output vsync; //VGA 场同步信号
output [3:0]red1;
output [3:0]green1;
output [3:0]blue1;
output [7:0] data_tube;//数码管段选数据
output [7:0] select_tube;//数码管位选

wire effect;//vga信号有效
wire up_key;//按键向上
wire down_key;//向下
wire [9:0]hc,vc;//640 480
wire [7:0] data_tx;//蓝牙输出
wire flag_bt;//蓝牙正在工作
wire stop;//游戏停止
wire up;//声音输出

key U1(clk,flag_bt,up,up_key,down_key);//按键

game
U2(data_tx,clk,flag_bt,effect,hc,vc,up_key,down_key,({red1,green1,blue1}),stop);//游戏

vga U3(clk,flag_bt,hsync,vsync,hc,vc,effect);//显示

bluetooth U4(clk,reset,data_rx,flag_bt,data_tx);//蓝牙

tube U5(clk,flag_bt,stop,data_tube,select_tube);//数码管

sound U6(clk,data_sound,up);//声音
```

```
endmodule
```

2.sound声音传感器模块

功能：得到声音信号，得到一个信号输出1，两个信号输出0

接口信号定义：



input clk;//时钟
input data_in;//声音信号输入
output up;//信号输出

代码：

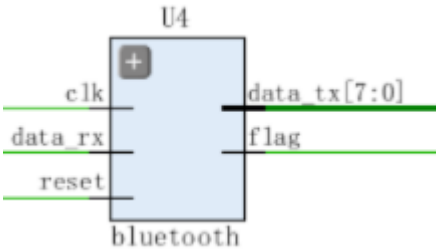
```
module sound(clk,data_in,up);
input clk;
input data_in;

output up;

//半秒缓冲
reg [31:0]cnt=0;
reg sound;
always @(posedge clk)
begin
    if(data_in==1&&cnt==0)
    begin
        sound=1;
        cnt=cnt+1;
    end
    else if(cnt!=0)
    begin
        cnt=cnt+1;
        sound=0;
        if(cnt==50000000)
        begin
            cnt=0;
        end
    end
```

```
        else
        begin
            sound=0;
            cnt=cnt;
        end
    end

    //向上
    reg up=0;
    always @(posedge sound)
        up<=~up;

endmodule
```

3.bluetooth蓝牙模块

功能：接受蓝牙信号，驱动其他模块工作，为游戏提供速度选择

接口信号定义：



input clk;//时钟
input reset;//重置
input data_rx;//信号输入
output reg flag;//驱动其他模块工作
output [7:0] data_tx;//游戏速度选择

代码：

```
    module bluetooth(clk,reset,data_rx,flag,data_tx);
    input clk;
    input reset;
    input data_rx;

    output reg flag;
    output [7:0] data_tx;

    parameter T = 15'd10414; //传输1bit所需计数值对应9600波特率

    //计数对应9600波特率
    reg start;
```

```
reg [14:0]cnt;
always @(posedge clk or posedge reset)
begin
   if(reset)
      cnt <= 15'd0;
   else if(cnt == T)
      cnt <= 15'd0;
   else if(start)
      cnt <= cnt + 1'b1;
   else
      cnt <= 1'b0;
end

//将采集数据的时刻放在波特率计数器每次循环计数的中间位置
wire collect;
assign collect = (cnt ==  15'd5208) ? 1'b1 : 1'b0;

//得到数据便产生下降沿
reg  [1:0] down;
always @(posedge clk or   posedge   reset)
begin
    if(reset)
        down <= 2'b11;
    else
    begin
        down[0]<=data_rx;
        down[1]<=down[0];
    end
end

//检测下降沿
wire nege_edge;
assign nege_edge= down[1]& ~down[0];

//UART协议
reg  [3:0]num;
reg rx_on;//接收字节时状态为1
always @(posedge clk or posedge reset)
```

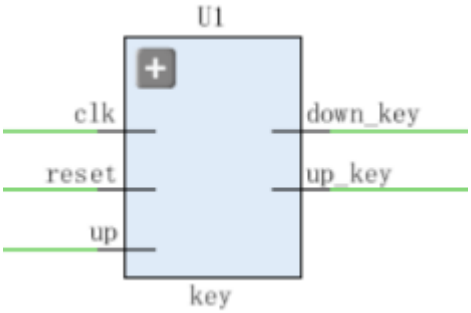
```
begin
    if(reset)
    begin
        start <= 1'b0;
        rx_on <= 1'b0;
    end
    else if(nege_edge)
    begin
        start <= 1'b1;
        rx_on <= 1'b1;
    end
    else if(num == 4'd10)
    begin
        start <= 1'b0;
        rx_on <= 1'b0;
    end
end

//存数据
reg  [7:0]rx_data_temp_r;//当前数据接收寄存器
reg  [7:0]rx_data_r;//用来锁存数据
always @(posedge clk or posedge reset)
begin
    if(reset)
    begin
        rx_data_r<= 8'd0;
        rx_data_temp_r<= 8'd0;
        num <= 4'd0;
    end
    else if(rx_on)
    begin
        if(collect)
        begin
            num <= num + 1'b1;
            case(num)
                4'd1: rx_data_temp_r[0] <= data_rx;
                4'd2: rx_data_temp_r[1] <= data_rx;
                4'd3: rx_data_temp_r[2] <= data_rx;
```

```
                4'd4: rx_data_temp_r[3] <= data_rx;
                4'd5: rx_data_temp_r[4] <= data_rx;
                4'd6: rx_data_temp_r[5] <= data_rx;
                4'd7: rx_data_temp_r[6] <= data_rx;
                4'd8: rx_data_temp_r[7] <= data_rx;
                default: ;
            endcase
        end
        else if(num == 4'd10)
        begin
            rx_data_r <= rx_data_temp_r;
            num <= 4'd0;
        end
    end
end

//数据传给输出
assign data_tx = rx_data_r;

//接受非0数据则flag=1
always @ (*)
begin
   if(data_tx!=8'b0011_0000&&data_tx!=8'b0)
   begin
       flag<=1;
   end
   else
   begin
       flag<=0;
   end
end

endmodule
```

4.key游戏按键模块

功能：得到sound模块的输出，为游戏提供up、down两个按键

接口信号定义：



input clk;//时钟
input reset;//重置
input up;//得到的声音输入
output reg up_key;//向上按键
output reg down_key;//向下按键

代码：

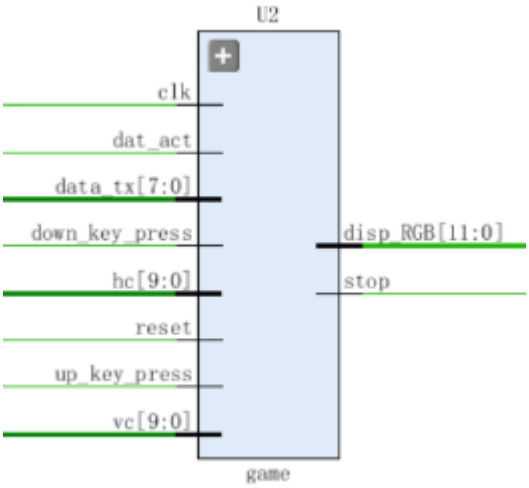
```
module key(clk,reset,up,up_key,down_key);
input clk;
input reset;
input up;

output reg up_key;
output reg down_key;

parameter T = 30'd1000_000;  //控制方块在y方向速度

//向上或者向下
reg [30:0] count1;
reg [30:0] count2;
always@(posedge clk or negedge reset)
begin
    if(!reset)
    begin
        count1 <= 0;
        count2 <= 0;
        up_key <= 0;
        down_key <= 0;
    end
    else
    begin
        if(up)
        begin
            if(count1 <= T)
            begin
                count1 = count1 + 1'b1;
                up_key <= 0;
```

```
            end
            else
            begin
                count1 <= 0;
                up_key <= 1;
            end
        end
        else  //下降
        begin
            if(count2 <= T)
            begin
                count2 = count2 +  1'b1;
                down_key <= 0;
            end
            else
            begin
                count2 <= 0;
                down_key <= 1;
            end
        end
    end
end

endmodule
```

5.game游戏模块

功能：作为游戏主体，实现游戏相关功能

接口信号定义：




```
input [7:0]data_tx;//控制挡板速度
input clk;//时钟
input reset;//重置
input dat_act;//VGA扫描有效
input [9:0]hc,vc;//VGA扫描
input up_key_press;//向上按键
input down_key_press;//向下按键
output [11:0]disp_RGB;//RGB输出
output reg stop;//游戏停止
```

代码：

```
module
game(data_tx,clk,reset,dat_act,hc,vc,up_key_press,down_key_press,di
sp_RGB,stop);
input [7:0]data_tx;
input clk;
input reset;
input dat_act;
input [9:0]hc,vc;
input up_key_press;
input down_key_press;

output [11:0]disp_RGB;
output reg stop;

//取蓝牙输出的后4位
wire [4:0]tx;
assign tx=data_tx[3:0];

//分频25MHz
reg cnt_clk=0;
reg vga_clk=0;
always @(posedge clk)
begin
    if(cnt_clk == 1)
    begin
        vga_clk <= ~vga_clk;
```

```
        cnt_clk <= 0;
    end
    else
        cnt_clk <= cnt_clk +1;
end

//开始画面
reg [18:0] addr = 0;
wire [11:0] rom_data;
reg [11:0]disp;

//ip核调用
big_rom me(.clka(cnt_clk), .addra(addr), .douta(rom_data));

//显示图片
always @ (posedge cnt_clk)
begin
    if (dat_act == 1)
    begin
        if (vc < 360 && vc > 120 && hc < 480 && hc > 160)
        begin
            addr <= (vc-120-1) * 320 + (hc-160) - 1;//通过vc、hc计
算出地址并获取图片对应位置RGB
            disp[11:8] <= rom_data[11:8];
            disp[7:4] <= rom_data[7:4];
            disp[3:0] <= rom_data[3:0];
        end
        else
        begin
            disp[11:8] <= 4'b1111;
            disp[7:4] <= 4'b1111;
            disp[3:0] <= 4'b1111;
        end
    end
    else
    begin
        disp[11:8] <= 0;
        disp[7:4] <= 0;
```

```
        disp[3:0] <= 0;
    end
end

//结束画面
reg [18:0] addr_over = 0;
wire [11:0] rom_data_over;
reg [11:0]disp_over;
//ip核调用
rom_over
you(.clka(cnt_clk), .addra(addr_over), .douta(rom_data_over));

//显示图片
always @ (posedge cnt_clk)
begin
    if (dat_act == 1)
    begin
        if (vc < 360 && vc > 120 && hc < 480 && hc > 160)
        begin
            addr_over <= (vc-120-1) * 320 + (hc-160) - 1;//通过vc、
hc计算出地址并获取图片对应位置RGB
            disp_over[11:8] <= rom_data_over[11:8];
            disp_over[7:4] <= rom_data_over[7:4];
            disp_over[3:0] <= rom_data_over[3:0];
        end
        else
        begin
            disp_over[11:8] <= 4'b1111;
            disp_over[7:4] <= 4'b1111;
            disp_over[3:0] <= 4'b1111;
        end
    end
    else
    begin
        disp_over[11:8] <= 0;
        disp_over[7:4] <= 0;
        disp_over[3:0] <= 0;
    end
```

```
end

//结束画面
reg [18:0] addr_bird = 0;
wire [11:0] rom_data_bird;
reg [11:0]data;
//ip核调用
bird
they(.clka(cnt_clk), .addra(addr_bird), .douta(rom_data_bird));

parameter border = 40;//定义正方形小块的边长
parameter ban = 20;//定义挡板的宽度
parameter long = 200;//定义挡板的长度
parameter magin = 160;//定义挡板的间隔
reg [10:0] push,push1,push2,push3;//VGA扫描，画出挡板和方块，并设置挡板移动的移动变量push
parameter move_x = 50; //方块的初始位置

//随机数
reg [7:0] rand_num;
parameter seed = 8'b1111_1111;
always@(posedge clk or negedge reset)
begin
    if(!reset)
        rand_num <= seed;
    else
    begin
        rand_num[0] <= rand_num[1] ;
        rand_num[1] <= rand_num[2] + rand_num[7];
        rand_num[2] <= rand_num[3] + rand_num[7];
        rand_num[3] <= rand_num[4] ;
        rand_num[4] <= rand_num[5] + rand_num[7];
        rand_num[5] <= rand_num[6] + rand_num[7];
        rand_num[6] <= rand_num[7] ;
        rand_num[7] <= rand_num[0] + rand_num[7];
    end
end
```

```
//随机数
wire [2:0]choose;
reg [8:0]type1;
assign choose = {rand_num[3],rand_num[6],rand_num[2]};

//随机数
always@(posedge clk )
begin
    case(choose)
        0:type1 = 0;
        1:type1 = 40;
        2:type1 = 80;
        3:type1 = 120;
        4:type1 = 160;
        5:type1 = 200;
        6:type1 = 240;
        7:type1 = 280;
        default: type1 = 280;
    endcase
end

//板块移动速度控制
reg move;
reg [32:0]counter;
reg [30:0]T_move;
always@(posedge clk or negedge reset)
begin
if(!reset)
begin
    T_move = 30'd10_000_00;
    counter <= 0;
    move <=0;
end
else
begin
    if(counter >= T_move)
    begin
        move = 1;
```

```
        if(T_move == 30'd100_000)
            T_move <=T_move;
        else
            T_move = T_move-10;
        counter = 0;
    end
    else
    begin
        move = 0;
        if(!stop)
            counter= counter + 1;
        else
            counter = 0;
    end
end
end

//板快位置
reg [8:0] rand0,rand1,rand2,rand3;
always@(posedge clk or negedge reset)
begin
if (!reset)
begin
    push<=640;  //初始位置设定
    push1 <= 640+ magin;
    push2 <= 640 + magin + magin;
    push3 <= 640 + magin + magin + magin;
end
else if (move)
begin
    if(push == 0)
    begin
        push <= 640;
        rand0 <=type1; //第一块板子的位置设定
    end
    else
    begin
        push <= push-tx;
```

```
    end
    if(push1 == 0)
    begin
        push1 <= 640;
        rand1 <=type1; //第二块板子的位置设定
    end
    else
    begin
        push1 <= push1-tx;
    end
    if(push2 == 0)
    begin
         push2 <= 640;
         rand2 <=type1; //第三块板子的位置设定
    end
    else
    begin
        push2<= push2-tx;
    end
    if(push3 == 0)
    begin
         push3 <= 640;
         rand3 <=type1;//第四块板子的位置设定
    end
    else
    begin
        push3 <= push3-tx;
    end
end
else
begin
    push <= push;
    push1 <= push1;
    push2 <= push2;
    push3 <= push3;
end
end
```

```
//游戏失败定义，方块与挡板"碰撞" 共设置四块挡板，四种情况
reg [9:0]move_y;
wire die1,die2,die3,die4;
assign die1=((rand0<move_y + border)&&(move_y < rand0+long)&&(push
< move_x+border) && (move_x < push + ban ));
assign die2=((rand1<move_y + border)&&(move_y < rand1+long)&&(push1
< move_x+border) && (move_x < push1 + ban ));
assign die3=((rand2<move_y + border)&&(move_y < rand2+long)&&(push2
< move_x+border) && (move_x < push2 + ban ));
assign die4=((rand3<move_y + border)&&(move_y < rand3+long)&&(push3
< move_x+border) && (move_x < push3 + ban ));

//游戏失败
wire false;
assign false = die1||die2||die3||die4;

//方块移动控制
always@(posedge clk or negedge reset)
begin
    if (!reset)
    begin
        move_y <= 240;
    end
    else if(stop)
    begin
        move_y<=move_y;
    end
    else if (up_key_press)
    begin
        if(move_y == 0)
        begin
            move_y <= move_y;
        end
        else
        begin
            move_y <= move_y-1'b1;
        end
    end
```

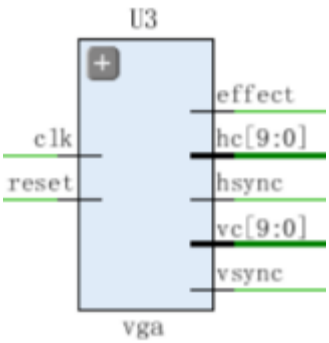
```
    else if (down_key_press)
    begin
        if(move_y>440)
        begin
            move_y <= move_y;
        end
        else
        begin
            move_y <= move_y+1'b1;
        end
    end
else
    move_y<=move_y;
end

//描述运动，"画图"

always@(posedge vga_clk or negedge reset)
begin
if(!reset)
begin
    data <= 0;
    stop <= 0;
end
else
begin
    if (hc>move_x
&&(hc<(move_x+border)&&(vc>move_y)&&(vc<move_y+border))) //小方块
    begin
        if(!false)
        begin
            addr_bird <= (vc-move_y-1) * 40 + (hc-move_x) - 1;//通
过vc、hc计算出地址并获取图片对应位置RGB
            data[11:8] <= rom_data_bird[11:8];
            data[7:4] <= rom_data_bird[7:4];
            data[3:0] <= rom_data_bird[3:0];
        end
        else
```

```
        begin
            data <= 12'b1111_0000_0000; //红色
            stop <=1;
        end
    end
    else if ((hc>push) && (hc<=push+ban) && (vc>=rand0) &&
(vc<=rand0+long))
    begin
        data <= 12'b0000_0000_1111;  //第一根横条
    end
    else if ((hc>push1) && (hc<=push1+ban) && (vc>=rand1) &&
(vc<=rand1+long))
    begin
        data <= 12'b0000_0000_1111;  //第二根横条
    end
    else if ((hc>push2) && (hc<=push2+ban) && (vc>=rand2) &&
(vc<=rand2+long))
    begin
        data <= 12'b0000_0000_1111;   //第三根横条
    end
    else if ((hc>push3) && (hc<=push3+ban) && (vc>=rand3) &&
(vc<=rand3+long))
    begin
        data <= 12'b0000_0000_1111;   //第四根横条
    end
else
    data <= 12'b1111_1111_1111;
end
end

//RGB显示选择
assign disp_RGB = reset ? (stop ? disp_over :(dat_act ?
data:12'b000000000000)): disp;

endmodule
```

6.VGA模块

功能：产生行扫描、场扫描、扫描有效信号

接口信号定义：




```
input clk;//时钟
input reset;//重置
output hsync;//行有效
output vsync; //场有效
output [9:0]hc ,vc;//行扫描、场扫描
output effect;//扫描有效
```

代码：

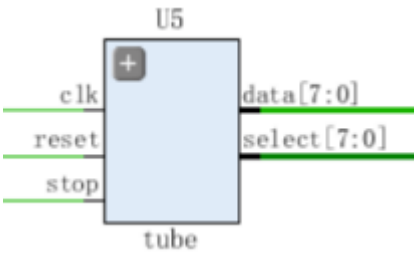
```
module vga(clk,reset,hsync,vsync,hc,vc,effect);
input clk;
input reset;

output hsync;
output vsync;
output [9:0]hc ,vc;
output effect;

parameter
hsync_end = 10'd95,
hdat_begin = 10'd143,
hdat_end = 10'd783,
hpixel_end = 10'd799,
vsync_end = 10'd1,
vdat_begin = 10'd34,
vdat_end = 10'd514,
vline_end = 10'd524;

//分频为25MHz
reg cnt_clk=0;
reg vga_clk=0;
always @(posedge clk)
begin
    if(cnt_clk == 1)
    begin
        vga_clk <= ~vga_clk;
```

```
        cnt_clk <= 0;
    end
    else
        cnt_clk <= 1;
end

//vga扫描
reg [12:0] hcount; //行扫描计数器
reg [12:0] vcount; //场扫描计数器
always @(posedge vga_clk)//行扫描
begin
if (hcount==hpixel_end)
    hcount <= 12'd0;
else
    hcount <= hcount + 12'd1;
end

assign hsync=(hcount<=hsync_end)?1'b0:1'b1;

always @(posedge vga_clk)//场扫描
begin
if(vcount==vline_end)
    vcount<=1'b0;
else if(hcount==hpixel_end)
    vcount<=vcount+1;
end

assign vsync=(vcount<=vsync_end)?1'b0:1'b1;

//信号有效
assign effect = ((hcount >= hdat_begin) && (hcount < hdat_end))&&
((vcount >= vdat_begin) && (vcount < vdat_end));

//计数器转成640 x 480
assign hc = hcount - hdat_begin;
assign vc = vcount - vdat_begin;

endmodule
```

7.tube数码管模块

功能：用数码管实现游戏时间记录

接口信号定义：




```
input clk;//时钟
input reset;//重置
input stop;//游戏停止
output [7:0] data;//数码管段选数据
output [7:0] select;//数码管位选
```

代码：

```
`timescale 1ns / 1ps
module tube(clk,reset,stop,data,select);
input clk;
input reset;
input stop;

output [7:0] data;//数码管段选数据
output [7:0] select;//数码管位选

wire [1:0] hour_h;
wire [3:0] hour_l;
wire [2:0] minute_h;
wire [3:0] minute_l;
wire [2:0] second_h;
wire [3:0] second_l;

clock
A1(clk,reset,stop,hour_h,hour_l,minute_h,minute_l,second_h,second_l
);

display
A2(clk,reset,hour_h,hour_l,minute_h,minute_l,second_h,second_l,data
,select);

endmodule
```

```
module display(clk,reset, hour_h, hour_l,
minute_h,minute_l,second_h,second_l,data,select);
input clk;
input reset;
input [1:0] hour_h;
input [3:0] hour_l;
input [2:0] minute_h;
input [3:0] minute_l;
input [2:0] second_h;
input [3:0] second_l;

output [7:0] data;
output [7:0] select;

parameter period= 100000;

//建立从0-9与数码管的映射关系
reg [6:0] Y_r;
reg [7:0] DIG_r;
reg [7:0] reg_a[9:0];
assign data ={1'b1,(~Y_r[6:0])};
assign select =~DIG_r;
initial
begin
    reg_a[0] <= 7'b0111111;
    reg_a[1] <= 7'b0000110;
    reg_a[2] <= 7'b1011011;
    reg_a[3] <= 7'b1001111;
    reg_a[4] <= 7'b1100110;
    reg_a[5] <= 7'b1101101;
    reg_a[6] <= 7'b1111101;
    reg_a[7] <= 7'b0100111;
    reg_a[8] <= 7'b1111111;
    reg_a[9] <= 7'b1100111;
end

//分频
reg [31:0]cnt;
```

```
reg clkout;
always @( posedge clk or negedge reset)
begin
    if (!reset)
        cnt <= 0 ;
    else
    begin
        cnt<= cnt+1;
        if (cnt == (period >> 1) - 1)
            clkout <= 1'b1;
        else if (cnt == period - 1)
        begin
            clkout <= 1'b0;
            cnt <= 1'b0;
        end
    end
end

//数码管每一位扫描
reg [2:0]scan_cnt=0 ;
always @(posedge clkout or negedge reset)
begin
    if (!reset)
        scan_cnt <= 0;
    else
    begin
        scan_cnt <= scan_cnt + 1;
        if(scan_cnt==3'd5)
            scan_cnt <= 0;
    end
end

//数码管选择
reg [3:0]N1,N2,N3,N4,N5,N6;
always @(scan_cnt)
begin
    N1<=second_l;
    N2<=second_h;
```

```
    N3<=minute_l;
    N4<=minute_h;
    N5<=hour_l;
    N6<=hour_h;
    case (scan_cnt)
        3'b000 : DIG_r <= 8'b0000_0001;
        3'b001 : DIG_r <= 8'b0000_0010;
        3'b010 : DIG_r <= 8'b0000_0100;
        3'b011 : DIG_r <= 8'b0000_1000;
        3'b100 : DIG_r <= 8'b0001_0000;
        3'b101 : DIG_r <= 8'b0010_0000;
        default :DIG_r <= 8'b0000_0000;
    endcase
end

//译码
always @ (scan_cnt)
begin
    case (scan_cnt)
        3'b000: Y_r = reg_a[N1];
        3'b001: Y_r = reg_a[N2];
        3'b010: Y_r = reg_a[N3];
        3'b011: Y_r = reg_a[N4];
        3'b100: Y_r = reg_a[N5];
        3'b101: Y_r = reg_a[N6];
        default: Y_r = 7'b0111111;
    endcase
end

endmodule

module clock(clk,
reset,stop,hour_h,hour_l,minute_h,minute_l,second_h,second_l);
input clk;
input reset;
input stop;

output [1:0]hour_h;
```

```
output [3:0]hour_l;
output [2:0]minute_h;
output [3:0]minute_l;
output [2:0]second_h;
output [3:0]second_l;

parameter  S=100000000;
parameter  M=60;
/*********************************************************/
//1秒计数器
reg [31:0] cnt;
always @(posedge clk or negedge reset)
begin
    if(!reset)
        cnt <= 15'd0;
    else if((cnt == S))
            cnt <= 15'd0;
        else
            cnt <= cnt + 1'b1;
 end

//开始跑
reg [1:0] reg1;
reg [3:0] reg2;
reg [2:0] reg3;
reg [3:0] reg4;
reg [2:0] reg5;
reg [3:0] reg6;
always @(posedge clk or negedge reset or posedge stop)
begin
if (!reset)
begin
    reg1 <= 2'd0;
    reg2 <= 4'd0;
    reg3 <= 3'd0;
    reg4 <= 4'd0;
    reg5 <= 3'd0;
    reg6 <= 4'd0;
```

```
end
else if(stop)
begin
    reg1 <= reg1;
    reg2 <= reg2;
    reg3 <= reg3;
    reg4 <= reg4;
    reg5 <= reg5;
    reg6 <= reg6;
end
else //时钟正常开始跑
begin
    if (cnt == S)  //一秒到了
    begin
        reg6 <= reg6 + 1'b1;
        if (reg6 == 4'd9)
        begin
            reg6 <= 4'd0;
            reg5 <= reg5 + 1'b1;
            if (reg5 == 3'd5)
            begin
                reg5 <= 3'd0;
                reg4 <= reg4 + 1'b1;
                if (reg4 == 4'd9)
                begin
                    reg4 <= 4'd0;
                    reg3 <= reg3 + 1'b1;
                    if (reg3 == 3'd5)
                    begin
                        reg3 <= 3'd0;
                        if (reg1 == 2'd2)
                        begin
                            reg2 <= reg2 + 1'b1;
                            if (reg2 == 4'd3)
                            begin
                                reg2 <= 4'd0;
                                reg1 <= 2'd0;
                            end
```

```
                        end
                        else
                        begin
                            reg2 <= reg2 + 1'b1;
                            if (reg2 == 4'd9)
                            begin
                                reg2 <= 4'd0;
                                reg1 <= reg1 + 1'b1;
                            end
                        end
                    end
                end
            end
        end
    end
end
end

//赋值给每一位
assign hour_h = reg1;
assign hour_l = reg2;
assign minute_h = reg3;
assign minute_l = reg4;
assign second_h = reg5;
assign second_l = reg6;

endmodule
```

五、测试模块建模

由于top模块不需要测试，game模块过于复杂，VGA模块显示的特殊性，因此这三个模块没有进行测试。

1. sound模块

```
`timescale 1ns / 1ps
module sound_tb();
reg clk;
reg data_in;
```

```
  wire up;
  sound uut(
  .clk(clk),
  .data_in(data_in),
  .up(up)
  );
  initial
  begin
      clk=0;
      data_in=0;
      #10 data_in=1;
      #500 data_in=0;
  end
  always
  begin
      #10 clk<=~clk;
  end
endmodule
```

2.bluetooth模块

```
  module tb_uart_rx;
    // Inputs
    reg clk;
    reg rst_n;
    reg data_rx;
    // Outputs
    wire [7:0] data_tx;
    wire rx_int;
    // Instantiate the Unit Under Test (UUT)
    uart_rx uut (
        .clk(clk),
        .rst_n(rst_n),
        .data_rx(data_rx),
        .data_tx(data_tx),
        .rx_int(rx_int)
    );
    always #10 clk = ~clk;
    initial begin
```

```
clk = 0;
rst_n = 0;
data_rx = 0;
// Wait 20 ns for global reset to finish
#20
rst_n = 1;
data_rx = 1;
#104160
rst_n = 1;
data_rx = 0;  //起始位：-1-0-
#104160          //传输11001011 (倒序)
rst_n = 1;
data_rx = 1;
#104160
rst_n = 1;
data_rx = 1;
#104160
rst_n = 1;
data_rx = 0;
#104160
rst_n = 1;
data_rx = 1;
#104160
rst_n = 1;
data_rx = 0;
#104160
rst_n = 1;
data_rx = 0;
#104160
rst_n = 1;
data_rx = 1;
#104160
rst_n = 1;
data_rx = 1;
#104160          //结束位 -0-1-
rst_n = 1;
data_rx = 0;
#104160
```

```
        rst_n = 1;
        data_rx = 1;
        #104160          //复位
        rst_n = 0;
        data_rx = 0;
    end
endmodule
```

3. key模块

```
`timescale 1ns / 1ps
module key_tb();
reg clk;
reg reset;
reg up;
wire up_key;
wire down_key;
key uut(
.clk(clk),
.reset(reset),
.up(up),
.up_key(up_key),
.down_key(down_key)
);
initial
begin
    clk=0;
    reset=0;
    up=0;
    #10 reset=1;
    #10 up=1;
    #2000 up=0;
    #2000 up=1;
end
always
begin
    #10 clk<=~clk;
end
```

```
endmodule
```

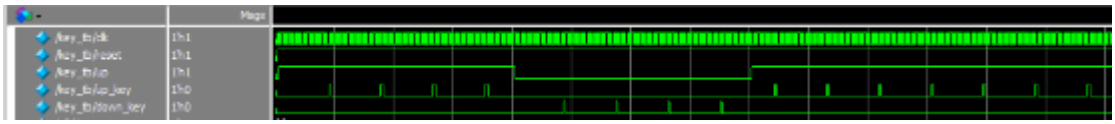
4.tube模块 clock 部分

```
`timescale 1ns / 1ps
module color_tb();
reg  clk;
reg rst_n;
wire [7:0] data;     //小时 十位
wire [7:0] select_wei;     //小时 个位
top uut(
.clk(clk),
.rst_n(rst_n),
.data(data),
.select_wei(select_wei)
);
initial
begin
rst_n=0;
clk=0;
#10 rst_n=1;
end
always
begin
#10 clk<=~clk;
end
endmodule
```

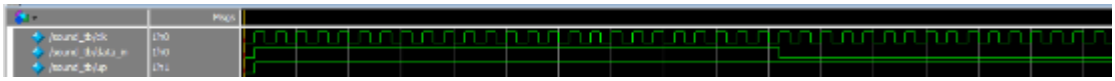
六、实验结果

1.Modelsim仿真波形图

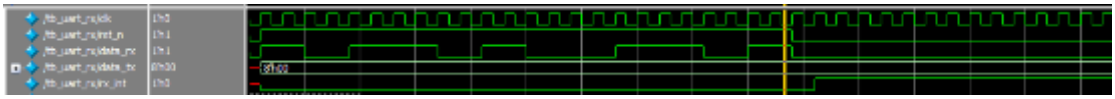
1.1key模块



1.2sound模块



1.3bluetooth模块



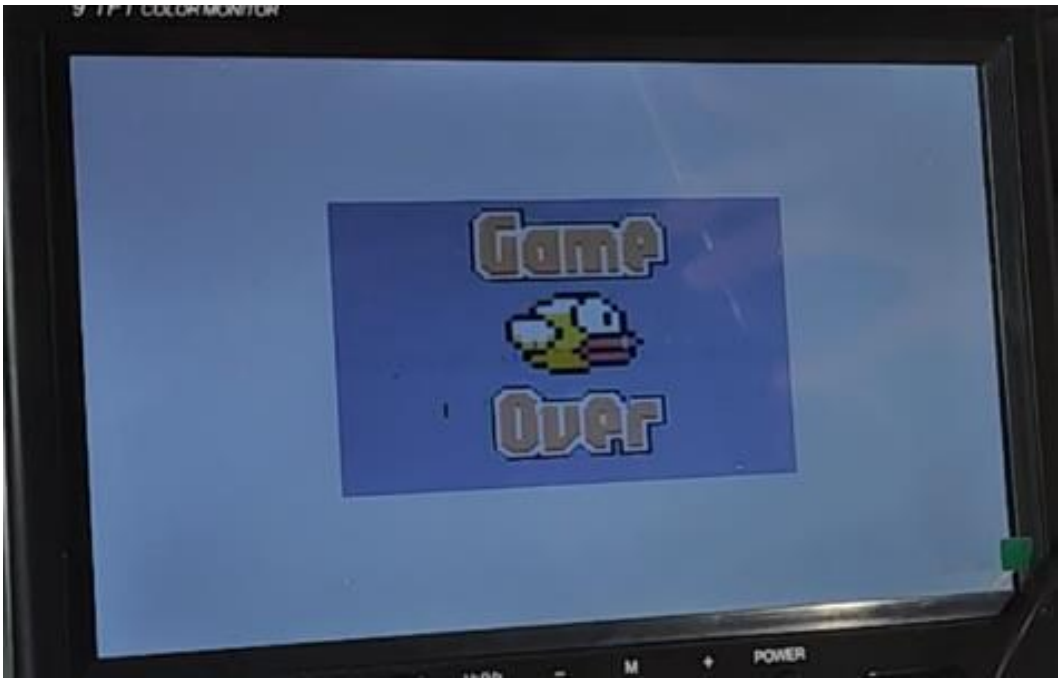
1.4tube模块 clock部分

2.实验结果贴图

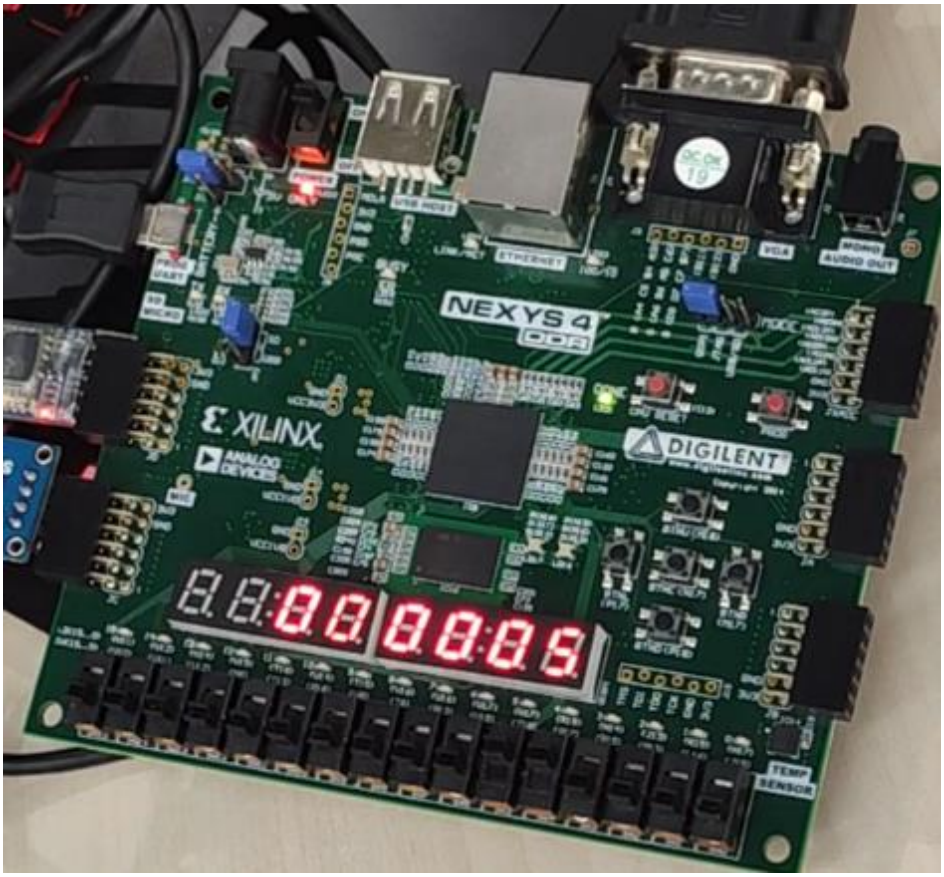
游戏开始

游戏中

游戏结束



数码管计时



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。因此数字系统是能够正常工作。蓝牙、声音传感器、VGA、开发板全部状态正常。但是，对于蓝牙的其他利用手段、声音传感器的频率信息，仍可以做进一步的开发。同时，对于其他外围部件在即时游戏中的应用也可以探索。

八、心得体会

1.课程收获：

本学期我们开设了数字逻辑课，在课上，我学到了很多在平时难以学到的东西。回顾这一学期的学习，感觉十分的充实，通过认真学习、亲自动手，使我进一步了解了数字逻辑的基本知识和设计方法，为我今后的学习奠定了良好的基础。

回顾学习过程，最直接的收获是了解了理论知识、提高了实验的基本操作能力，并对开发板、外围部件有了些了解，在最后的综合实验中，我更是受益匪浅。但感到更重要的收获是培养了自己对数字逻辑的兴趣。我深深体会到，

学习不单单要将理论知识学扎实了，也要注重实际动手操作能力，学完了课本知识，加上随后的实验，我感觉学到了很多知识，并且在实践中加深对书本上的理论知识的理解。

2.课程建议：

2.1 希望老师能继续采取多样形式和学生互动（提问，分组讨论等），督促学生积极学习。

2.2 上实验课时，希望老师多阐明实验原理。并且结合课本，举出实例。

2.3 希望大作业能提供外围部件的中文详细资料。

3.数字芯片设计之我见：

作为被“卡脖子”的重要技术之一，中国芯片的发展近年来饱受各界的密切关注。

中国大部分芯片企业创新能力不足

在摩尔定律的作用下，半导体制造技术进步呈指数级，芯片性能提升简单直接。近年来新一代工艺产品的上市时间很大程度上决定了芯片产品的生死，导致许多的细致设计优化工作都只能停留在学术圈。但随着后摩尔时代的来临，再想获得半导体性能的提升必须精耕细作，依靠设计，通过芯片架构、算法、精度等维度的优化，从软件到硬件进行全线优化，而且物联网和人工智能也对芯片设计提出了新的需求。

当前中国芯片产业正面临着诸多挑战，如产品无法满足市场的需求、技术路线仍在跟随、高端通用芯片领域无法直接参与竞争等。同时，我国设计企业还处于国际巨头后面亦步亦趋的状态，依赖工艺和EDA工具进步来实现产品的更新换代，大多数企业的创新能力不够，只能停留在低端的陷阱里无法自拔。而且由于差距较大，无法在市场上同国际巨头直接竞争，不得不将主攻方向转向特定市场，这将影响我国为数不多的高端通用芯片公司的发展。

天时地利人和的发展良机

尽管面临种种困难，但是，如今的中国芯片产业正迎来了天时地利人和的发展良机。天时即摩尔定律走向终结，行业迎来新生，创新变得非常重要；地利即这一次的发展机会属于物联网，大数据和人工智能，而这三个方面的最大的需求就在中国。目前国内集成电路设计产业已经有了一些很好的团队，而且一批有既有产业经验，也有管理能力的精英也逐步归国，越来越多投资机构开始关注芯片产业——这便是人和。

持续创新 瞄准细分市场

目前，中高端产品占了整个芯片市场的绝大部分，而低端芯片仅仅占了很少的一部分。很多中国芯片公司缺少积累，做不了中高端产品，只能紧盯某个低端产品试图用低价抢占市场。或者由于没有持续创新的能力，产品上市挣了

一笔后，毛利率回落，又逐渐陷入低价竞争的泥潭。譬如端芯片通常为SOC（系统级芯片），而设计SOC是一个系统工程，需要成建制的团队，还需要购买或者自己研制大量的IP核。每一家创新的芯片设计公司，都应该有自己特有的IP，这才是芯片公司的核心价值。

同时，瞄准细分市场对中国的初创企业也非常重要。因为通用处理器虽然销量很大，但全球处理器芯片巨头的产品优势太大，国内企业想与之竞争非常困难。而细分市场比如指纹识别芯片，虽然市场小，但机会很大。国际上的巨头公司通常会瞄准一年10亿美金以上营收的产品做设计。对这些巨头而言，一年1亿美金的产品没有什么意义，他们不会去做细分品类而只会做通用产品，但这恰恰是中国初创企业的机会。因为摩尔定律已经走到头了，功耗的限制、场景的需求、AI技术的突破，会带来一系列的细分场景。每个场景都有独特需求，国内市场可能都不止1亿美金，这便能成就一家上市公司。而国际大公司往往很难抓住这些领域——因为客户大多都在中国。中国的下游厂商需求变化快，软件技术支持也要求非常及时，因此国外大公司其实难以与本土企业竞争。

尤其应当引起投资者注意的是，不同于互联网创业的个人英雄式领导，芯片团队的经验和完整性是重要的投资评估要素。一个没有流片经验的团队没法顺利完成量产，芯片设计公司的核心是设计人员。首先要有一个好的带头人，一流的设计汇报给二流设计人员是很难长期维持的，创业团队中的技术负责人一定要足够优秀。在后摩尔定律时代，评估数字芯片公司的团队不能只看前后端芯片设计人员，更要看软件与架构团队的完整性——软硬件协同设计的能力通常决定了产品的价值。随着技术迭代速度加快，RISC-V等开源进入了芯片领域，团队中是否有站得高、看得远，能够引领技术路线的带头人也很重要。创业者要培育核心竞争力，而核心的竞争力来自于持续创新，竞争优势不是核心竞争力。第一难以长久，需要有差异化，做细分领域的唯一