The Design of the Responder

Development Plan and Execution Records

1. Group Work Distribution

11811620毛尊尧：Display，Score

11810614刘洺琛：Control, Timer, Buzzer with different songs

11810206孙涛：Compere, Responder, Setting, Retrieve（This is what we improved after the class presentation. Though we have only successfully finished the design and simulation part, we will finish it in future.）

2. Execution Records

Time	Task	Responsible	Implementation
2019.12.12	Module division and work distribution	刘洺琛	Well done
2019.12.16	Discuss and show the work having been done	毛尊尧	Well done
2019.12.19	First time to merger and solve problems	孙涛	Well done
2019.12.21	Successfully merge and try to optimize	毛尊尧	Well done
2019.12.22	Find problems and solve them	孙涛	Well done
2019.12.24	Achieve special functions and try to improve	刘洺琛	Well done

Demand Analysis

1. Timer:

It is a counter that minus 1 at one clock period. It also can stopped when the stoptime signal is triggered.

2. Score part:

Storage the grade of 4 players, and should change the score when certain signals are triggered.

3. Responder part:

read the signal from player buttons and transfer the signals to score part and timer part.

4. Showing part:

Read the mode signal from control part and read the data from other parts.present the corresponding information on 7-segment tube.

5. Control part:

This is a finite state machine, when when certain button is pushed, it can change change the state, and let the digital tube present the current mode contents.(timing, score,ready,winner,etc.)

6. Song player:

Play background music when certain signal is triggered.

How to work

Press the RST, display the GO

After the host press the starttimer, start time, at the same time a buzzer sound

Someone vies to answer first, display vies to answer first Numbers and the score:

Press Yes/NO, add/reduce his score

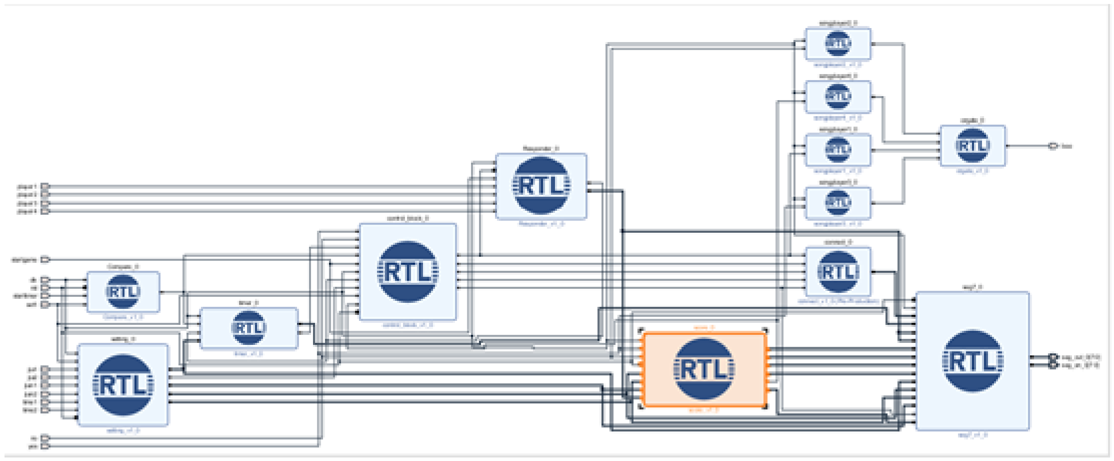
Press the startgame, into the next question, again according to GO

Press set, enter setup mode, display the longest vies to answer first time, the number of games, plus right and wrong points:

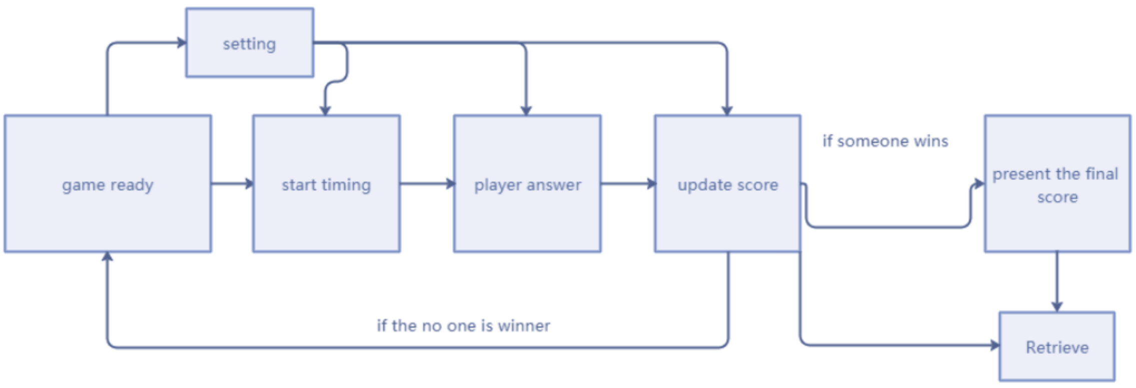
Off the set, to return to Go interface:

Game over(one win at least 10 scores), cycle show each score:

Then show the winner number and score



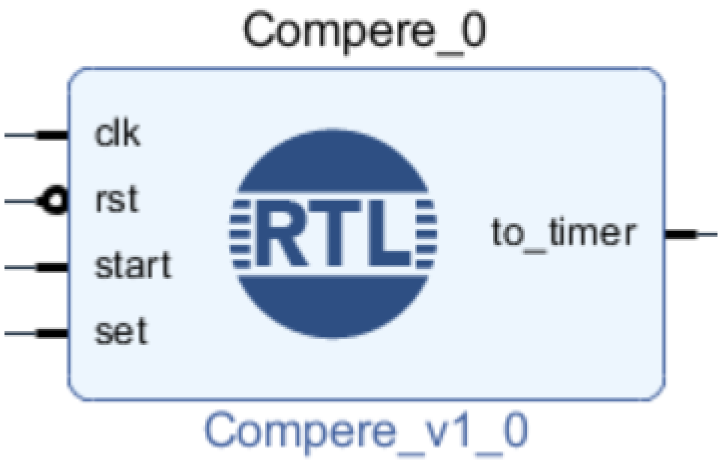
System Execution Process



Modules

1. Compere

Port description



Input:

Start: start the timer

Set: start the setting mode

Output :

to_timer: transfer the start signal to timer

Design idea:

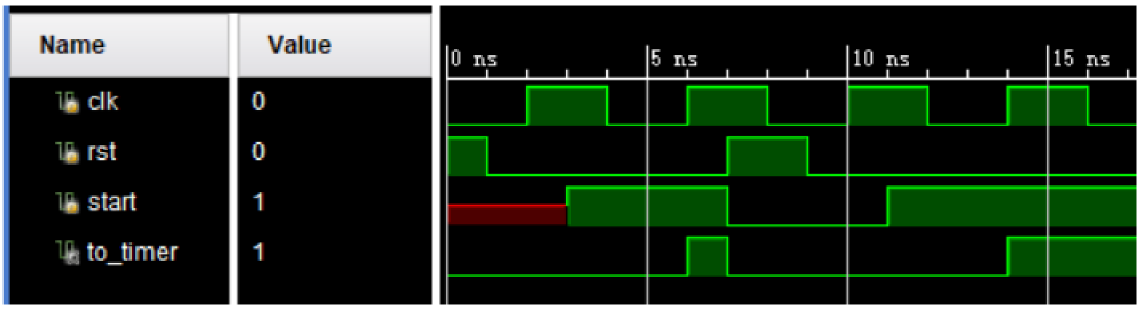
This part is used to control the beginning of the game.

```
module Compere(input clk,rst,start,output reg to_timer);
always @(posedge clk,posedge rst) begin
    if(rst)
    begin
    to_timer<=1'b0;
    end
else begin
    if(start)
    begin
        to_timer<=1'b1;
    end
    else
        to_timer<=1'b0;
end
end
endmodule
```

Simulation:

```
module compere_sim();
reg clk,rst,start;
wire to_timer;
Compere u(clk,rst,start,to_timer);
always #2 clk=~clk;
```

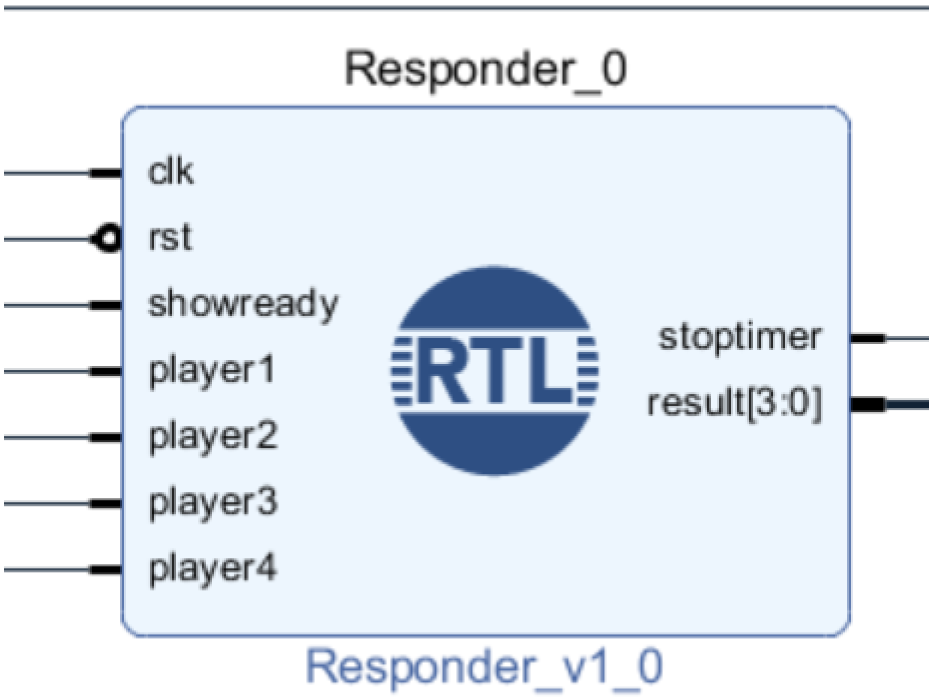
```
initial fork
rst=1;clk=0;
    #1 rst=1;
    #1 rst=0;
    #3 start=1;
    #7 rst=1;
    #7 start=0;
    #9 rst=0;
    #11 start=1;
    #40 $finish;
 join
endmodule
```



Simulation results: When the rst is invalid, the to_timer changes to 1 at the posedge of clk if start is 1.

2. Responder

Port description



Input:

Showready: when finishing one turn(after judging the correctness of one player) ,start a new turn.

Player1-player4: the answer button.

Output:

Stop timer: when someone pushes the button, transfer a signal to timer

Result: transfer the player pushes the button first.

Design idea:

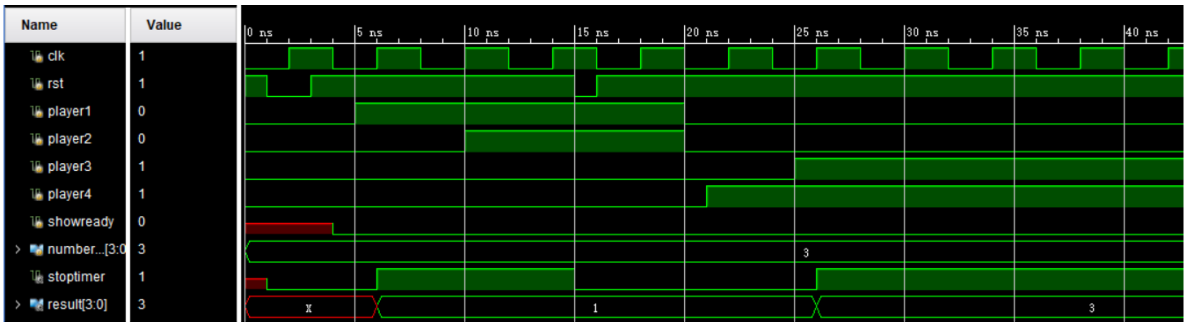
Using combinatorial logic to make the first one answered is valid and if any one else answer, the result will not change.

```
module Responder(input clk,rst,showready,player1,player2,player3,player4,
[3:0] number_of_player, //人数范围2-4, 人数用一位BCD码表示
output reg stoptimer,reg[3:0] result);
always@(posedge clk,negedge rst)begin
if(~rst)  //异步复位信号rst进行初始化
    begin
    stoptimer<=1'b0;
    end
else if(showready) result<=4'b0000;
else begin if(stoptimer==0)
begin //使用组合逻辑仅记录第一个有效抢答的结果,使用枚举法可实现人数2-4人参加
if( number_of_player==4)begin
    case({player1,player2,player3,player4})
        4'b1000: begin result= 4'b0001;stoptimer=1'b1;end
        4'b0100: begin result= 4'b0010;stoptimer=1'b1;end
        4'b0010: begin result= 4'b0011;stoptimer=1'b1; end
        4'b0001: begin  result= 4'b0100;stoptimer=1'b1; end
        default:  result=result ;
    endcase
end
else if( number_of_player==3)begin
      case({player1,player2,player3,player4})
          4'b1000:begin result= 4'b0001;stoptimer=1'b1; end
          4'b0100:begin result= 4'b0010;stoptimer=1'b1;  end
          4'b0010: begin result= 4'b0011;stoptimer=1'b1;end
          4'b1001: begin result= 4'b0001;stoptimer=1'b1;end
          4'b0101: begin result= 4'b0010;stoptimer=1'b1; end
          4'b0011: begin result= 4'b0011;stoptimer=1'b1; end
          default: result=result;
      endcase
end
else if( number_of_player==2) begin
      case({player1,player2,player3,player4})
          4'b1000:begin result= 4'b0001;stoptimer=1'b1; end
          4'b0100:begin result= 4'b0010;stoptimer=1'b1; end
          4'b1001:begin result= 4'b0001;stoptimer=1'b1; end
          4'b0101: begin result= 4'b0010;stoptimer=1'b1; end
          4'b1010: begin result= 4'b0001;stoptimer=1'b1; end
          4'b0110: begin result= 4'b0010;stoptimer=1'b1; end
          4'b1011: begin result= 4'b0001;stoptimer=1'b1; end
          4'b0111: begin result= 4'b0010;stoptimer=1'b1; end
          default: result=result;
      endcase
end
end
end
end
```

```
endmodule
```

Simulation:

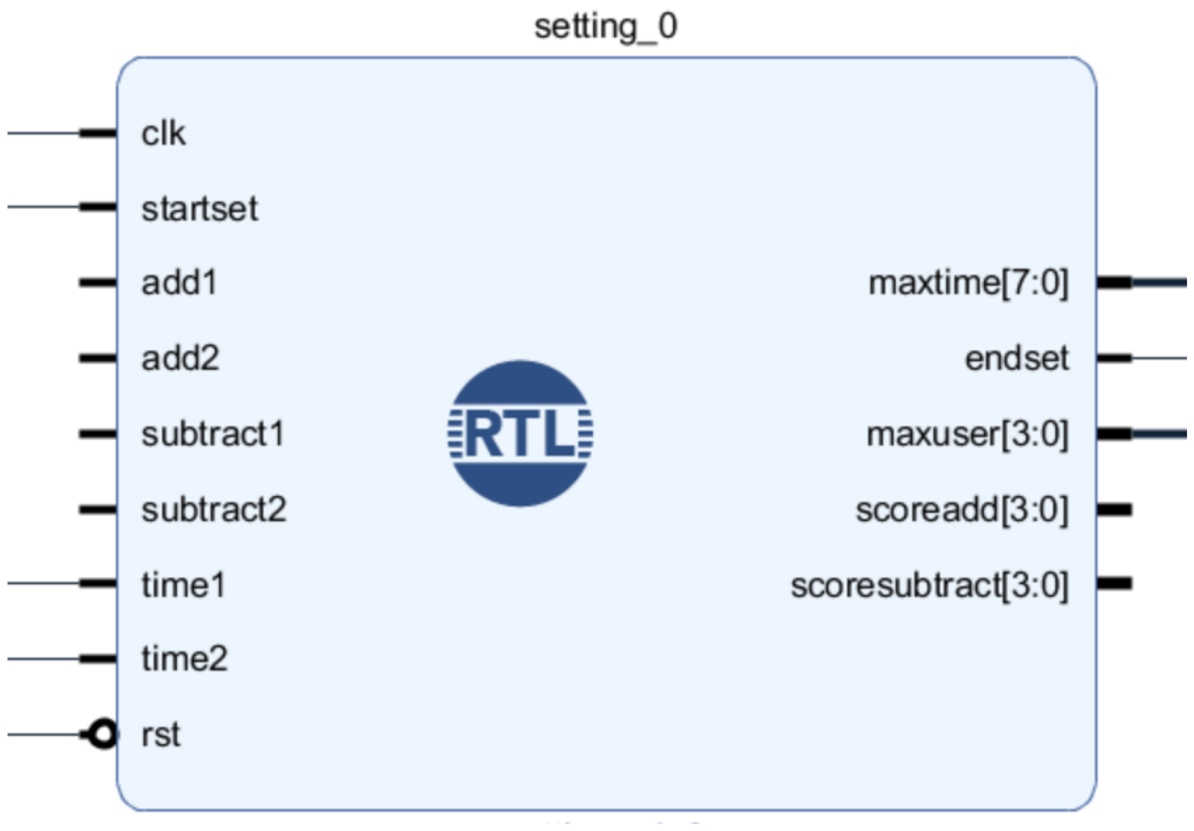
```
module responder_sim();
reg clk,rst,player1,player2,player3,player4,showready;
reg [3:0]number_of_player;
wire stoptimer;
wire [3:0] result;
Responder u(clk,rst,showready,player1,player2,player3,player4,
             number_of_player,stoptimer,result);
always #2 clk=~clk;
initial fork
clk=0;
rst=1;
number_of_player=4'b0011;
{player1,player2,player3,player4}=4'b0;
    #1 rst=0;
    #3 rst=1;
    #4 showready=0;
    #5 player1=1;
    #10 player2=1;
    #15 rst=0;
    #16 rst=1;
    #20 player1=0;
    #20 player2=0;
    #21 player4=1;
    #25 player3=1;
 join
endmodule
```



From the waveform diagram, we can see that in the first round, the player 1 first picked up, the player 2 then picked up, and the result showed player 1. In the second round, because the number of players in the game is set to 3, Player 4 has no effect. Player 3 gets the answer first, and the result shows that player 3. Simulation results meet our expectations.

3. Setting

Port description



Input:

Start set: enable the settings

add1\add2: control the add value

subtract1\subtract2:control the minus value

Time1\time2:control the timing value

Rst: reset the value to the default value

Output:

Maxtime: to timer

Endset: to control

Maxuser: to score

scoreadd\scoresubtract: the current add value and minus value

Design idea:

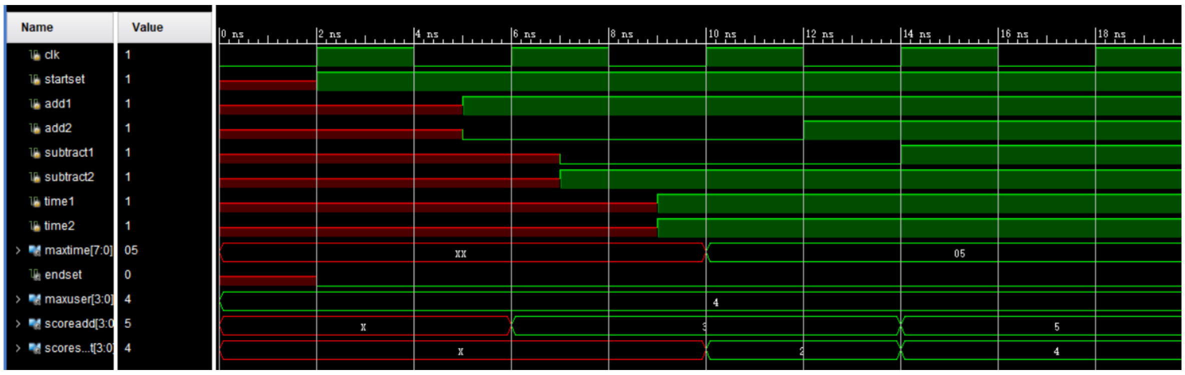
The module is used to set the status in the competition.

```
module setting(
input clk,startset,input add1,input add2,input subtract1,input subtract2,
input time1,input time2,
output wire[7:0]maxtime,wire endset,wire[3:0] maxuser,wire[3:0] scoreadd,
       wire [3:0] scoresubtract
);
   reg [7:0]maxtime1;
   reg [3:0]scoreadd1;
   reg [3:0]scoresubtract1;
```

```
    reg [3:0]maxuser1=4'b0100;
    assign maxtime=maxtime1;
    assign scoreadd=scoreadd1;
    assign scoresubtract =scoresubtract1;
    assign maxuser=maxuser1;
    always@(posedge clk)
    case({add1,add2})
    2'b00:scoreadd1=4'b0001;
    2'b01:scoreadd1=4'b0010;
    2'b10:scoreadd1=4'b0011;
    2'b11:scoreadd1=4'b0101;
    endcase
    always@(posedge clk)
    case({subtract1,subtract2})
    2'b00:scoresubtract1=4'b0001;
    2'b01:scoresubtract1=4'b0010;
    2'b10:scoresubtract1=4'b0011;
    2'b11:scoresubtract1=4'b0100;
    endcase
    always@(posedge clk)
    case({time1,time2})
    2'b00:maxtime1=8'b00110000;
    2'b01:maxtime1=8'b00010000;
    2'b10:maxtime1=8'b01000000;
    2'b11:maxtime1=8'b00000101;
    endcase
    assign endset=~startset;
endmodule
```

Simulation:

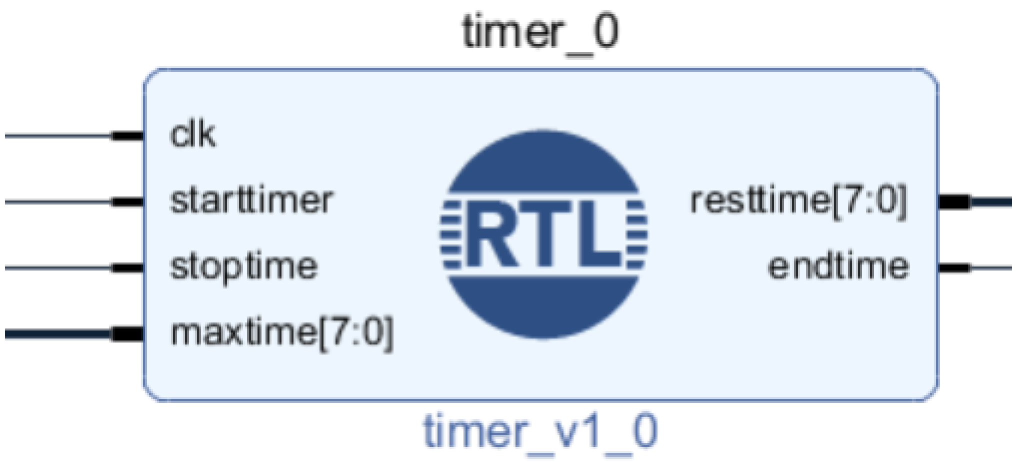
```
module test_setting();
reg clk,startset,add1,add2,subtract1,subtract2,time1,time2;
wire [7:0]maxtime;
wire endset;
wire[3:0] maxuser;
wire[3:0] scoreadd;
wire [3:0] scoresubtract;
setting
u(clk,startset,add1,add2,subtract1,subtract2,time1,time2,maxtime,endset,maxuser,
scoreadd,scoresubtract);
always #2clk=~clk;
    initial fork
    clk=0;
    #2startset=1;
    #5add1=1;
    #5add2=0;
    #7subtract2=1;
    #7subtract1=0;
    #9time1=1;
    #9time2=1;
    #11time2=1;
    #12add2=1;
    #14subtract1=1;
 join
endmodule
```



We can see that the simulation results are in line with expectations. At every posedge of the clock the circuit will change the value of the output according to the input, and then set the game state.

4. Timer

Port description



Input:

Clk: the system clock

Starttimer: start the game

Stoptime: when one player push buttons, the timing will end.

Maxtime:read the data from settings.

Output:

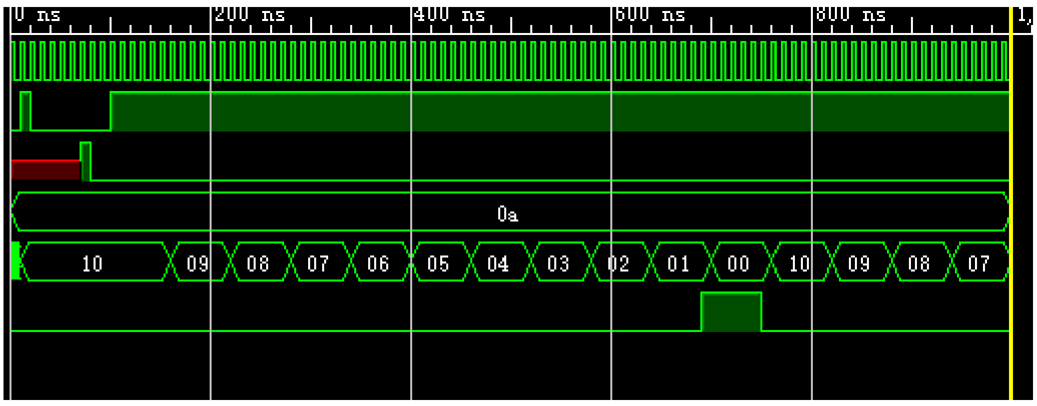
Resttime:to showing part, present the rest time(8bit represent two BCD numbers)

Design idea:

```
module timer(
    input clk,starttimer,stoptime,input[7:0]  maxtime,
    output reg[7:0]resttime,wire endtime
    );
    reg [7:0]count;
    reg [39:0]t;
    reg clk1;
    assign endtime=(count===0);
    always@(posedge clk)
    begin   {resttime[7],resttime[6],resttime[5],resttime[4]}<=count/10;
  {resttime[3],resttime[2],resttime[1],resttime[0]}<=count%10;
    end
    always@(posedge clk)
    begin
    if(!starttimer) begin t=40'b0;clk1=1;end
    t<=t+1;
    if(t==50000000)begin t<=0; clk1=~clk1;end//be changed in Simulation
       end
     always@(posedge clk1)
     if(!starttimer)
     begin
     count<={maxtime[7],maxtime[6],maxtime[5],maxtime[4]}*4'b1010+
{maxtime[3],maxtime[2],maxtime[1],maxtime[0]};
      end
     else if(count==7'b0000000||stoptime)
     begin
     count<={maxtime[7],maxtime[6],maxtime[5],maxtime[4]}*4'b1010+
{maxtime[3],maxtime[2],maxtime[1],maxtime[0]};
      end
     else
     begin
     count<=count-1;
     end
endmodule
```

Simulation

```
module s();
    reg clk,starttimer,stoptime;reg[7:0]  maxtime;
             wire [7:0]resttime;wire endtime;
    timer u1(.clk(clk),.starttimer(starttimer),.stoptime(stoptime),
                 .maxtime(maxtime),.resttime(resttime),.endtime(endtime));
    always
    begin;
    #5 clk=~clk;
    end
    initial
    begin
    clk<=1;
    maxtime<=10;starttimer<=0;
    #10 starttimer<=1;
    #10 starttimer<=0;
    #50 stoptime<=1;
    #10 stoptime<=0;
    #10 starttimer<=0;
    #10 starttimer<=1;
    #10 starttimer<=1;
    end
endmodule
```

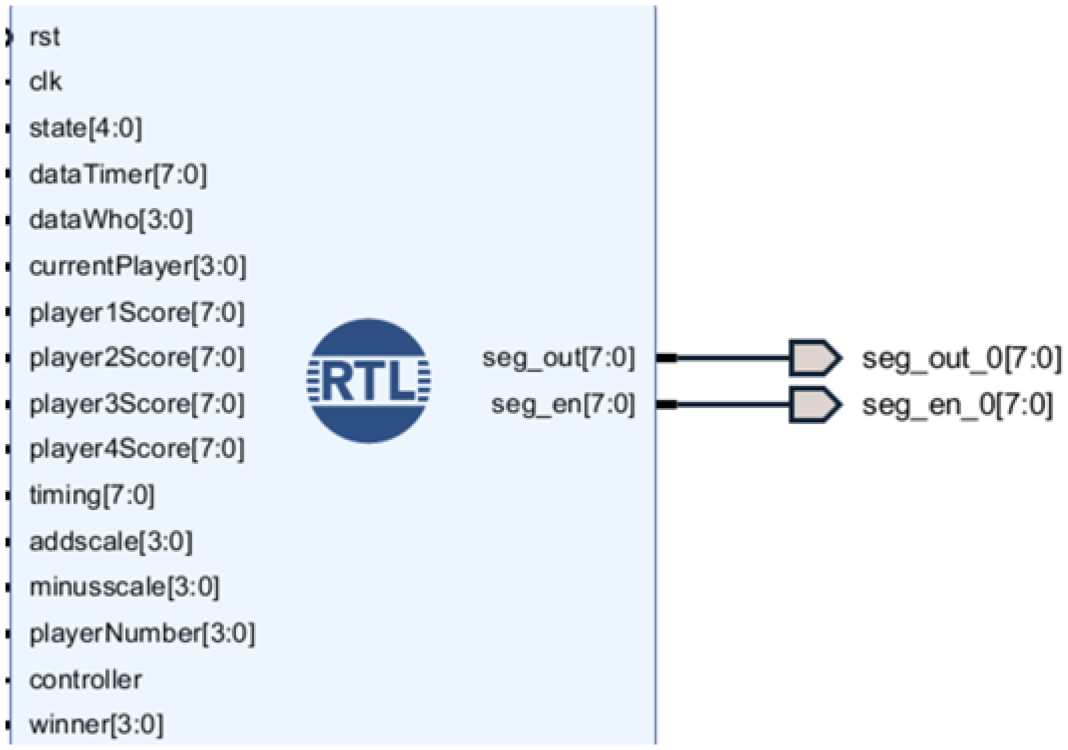


(maxtimer=10)

Time run out or one player press the button(stoptimer=1,stop reduce the rest time),there will be a endtimer,which is equal to 1.

5. Show_part

Port description



intput:

clk: the system clock

state:read the present mode and present different number

datatimer:read the timer number

datawho:read the current player data

currentplayer: test the current player

player1Score-player4Score:read the data from score part

timing\addscale\minusscale\playerNumber:read the data from setting part

winner:read the winner information from score part

output:

seg_en\seg_out: the 7 segment digital tube output.

Design idea:

```
module seg7(input rst,
            input clk,
            input [4:0]state,
            input [7:0] dataTimer,
            input [3:0] dataWho,
            input [3:0] currentPlayer,
            input [7:0] player1Score,
            input [7:0] player2Score,
            input [7:0] player3Score,
            input [7:0] player4Score,
            input [7:0] timing,
            input [3:0] addscale,
            input [3:0] minusscale,
            input [3:0] playerNumber,
            input  controller,
            input [3:0]winner,
            output [7:0]seg_out,
            output [7:0] seg_en
             );
           reg showfinal=0;
            reg [4:0]clk_out=5'b00000;
            reg [40:0]cnt=0;
              reg [3:0]lightflow=4'b0000;
            always@ (posedge clk or posedge rst)                    //结束状
态的
            begin
                if(rst)
                begin
                cnt<=0;
                clk_out<=0;
                showfinal<=0;
                lightflow=4'b0000;
                end
                else if(cnt==2)begin

                clk_out<=clk_out+1;
                cnt<=0;
                end
                else begin
                cnt<=cnt+1;
                end
            end
reg [7:0]decimalpoint=8'b00000000;
wire [7:0]decimalCache;
 reg tick=0;
reg [30:0]counter=0;
reg [3:0]segLocation=0;
reg [3:0] num=0;
reg en_point=8'b11111111;
reg [3:0]currentPlayetScore;
reg [7:0] bit_control;
reg [3:0] bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0;
```

```
reg[7:0] decimal_p=8'b00000000;
scan_show
show(clk,bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0,bit_control,decimal_p,seg_en,
seg_out);
reg [3:0]one,ten;
wire [3:0] player1ScoreTen;                         //将两位bcd码转换为数字
wire [3:0] player1ScoreOne;
assign player1ScoreTen=
{player1Score[7],player1Score[6],player1Score[5],player1Score[4]};
assign player1ScoreOne=
{player1Score[3],player1Score[2],player1Score[1],player1Score[0]};
wire [3:0] player2ScoreTen;
wire [3:0] player2ScoreOne;
assign player2ScoreTen=
{player2Score[7],player2Score[6],player2Score[5],player2Score[4]};
assign player2ScoreOne=
{player2Score[3],player2Score[2],player2Score[1],player2Score[0]};
wire [3:0] player3ScoreTen;
wire [3:0] player3ScoreOne;
assign player3ScoreTen=
{player3Score[7],player3Score[6],player3Score[5],player3Score[4]};
assign player3ScoreOne=
{player3Score[3],player3Score[2],player3Score[1],player3Score[0]};
wire [3:0] player4ScoreTen;
wire [3:0] player4ScoreOne;
assign player4ScoreTen=
{player4Score[7],player4Score[6],player4Score[5],player4Score[4]};
assign player4ScoreOne=
{player4Score[3],player4Score[2],player4Score[1],player4Score[0]};
always @(posedge clk)
begin
case(state)
    5'b00001:
    //开始状态，显示“GO”
    begin
    bit_control<=8'b11000000;
   {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'b1111,4'd0,4'd3,4'd4,4'd5,4'd6,4'd7,4'd8};
   end
      5'b00010:
      //结束状态，滚动显示每位玩家的得分，及最终冠军
begin
               case(clk_out)
                     5'b00000:
                        begin
                        bit_control<=8'b00100011;//1
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd1,4'd1,4'd1,4'd1,player1ScoreTen,player1ScoreOne,player1ScoreTen,player1Scor
eOne};
                        end
                    5'b00001:
                        begin
                        bit_control<=8'b01000110;//2
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd1,4'd1,4'd1,4'd1,player1ScoreTen,player1ScoreTen,player1ScoreOne,player1Scor
eOne};
                        end
                    5'b00010:
                        begin
                        bit_control<=8'b10001100;//3
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd1,4'd1,4'd1,4'd1,player1ScoreTen,player1ScoreOne,player1ScoreTen,player1Scor
eOne};
                        end
                     5'b00011:
                        begin
                        bit_control<=8'b00100011;//4
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd2,4'd2,4'd2,4'd2,player2ScoreTen,player2ScoreOne,player2ScoreTen,player2Scor
eOne};
                        end
                      5'b00100:
                         begin
                        bit_control<=8'b01000110;//5
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd2,4'd2,4'd2,4'd2,player2ScoreTen,player2ScoreTen,player2ScoreOne,player2Scor
eOne};
                        end
                       5'b00101:               //6
                         begin
                        bit_control<=8'b10001100;
                        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd2,4'd2,4'd2,4'd2,player2ScoreTen,player2ScoreOne,player2ScoreTen,player2Scor
eOne};
                        end
                       5'b00110:          //7
                           begin
                           bit_control<=8'b00100011;
                           {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd3,4'd3,4'd3,4'd3,player3ScoreTen,player3ScoreOne,player3ScoreTen,player3Scor
eOne};
                           end
                        5'b00111:        //8
                              begin
                              bit_control<=8'b01000110;
                              {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd3,4'd3,4'd3,4'd3,player3ScoreTen,player3ScoreTen,player3ScoreOne,player3Scor
eOne};
                              end
                         5'b01000:        //9
                                 begin
                                 bit_control<=8'b10001100;
                                 {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd3,4'd3,4'd3,4'd3,player3ScoreTen,player3ScoreOne,player3ScoreTen,player3Scor
eOne};
                                 end
                          5'b01001:     //10
                                    begin
                                    bit_control<=8'b00100011;
                                    {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd4,4'd4,4'd4,4'd4,player4ScoreTen,player4ScoreOne,player4ScoreTen,player4Scor
eOne};
                                    end
                          5'b01010:    //11
                                       begin
                                       bit_control<=8'b01000110;
```

```
{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd4,4'd4,4'd4,4'd4,player4ScoreTen,player4ScoreTen,player4ScoreOne,player4Scor
eOne};
                                       end
                             5'b01011:   //12
                                          begin
                                          bit_control<=8'b10001100;

 {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{4'd4,4'd4,4'd4,4'd4,player4ScoreTen,player4ScoreOne,player4ScoreTen,player4Scor
eOne};
                                          end


         default:
           begin
                              bit_control<=8'b10001111;
                              {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{winner,4'b0,4'b0,4'b0,4'd6,4'd6,4'd6,4'd6}; end
        endcase
end
        5'b00100:            //设置状态
        begin
          bit_control<=8'b11010101;
           {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}<=
{{timing[7],timing[6],timing[5],timing[4]},
{timing[3],timing[2],timing[1],timing[0]},4'd0,addscale,4'd0,minusscale,4'd0,pla
yerNumber};
        end
         5'b01000:                            //计分状态，显示当前玩家及其的房

                        begin
                        bit_control=8'b10000011;
                        case(currentPlayer)
                        4'b0001:{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{currentPlayer,4'b0,4'b0,4'b0,4'b0,4'b0,
{player1Score[7],player1Score[6],player1Score[5],player1Score[4]},
{player1Score[3],player1Score[2],player1Score[1],player1Score[0]}};
                        4'b0010:{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{currentPlayer,4'b0,4'b0,4'b0,4'b0,4'b0,
{player2Score[7],player2Score[6],player2Score[5],player2Score[4]},
{player2Score[3],player2Score[2],player2Score[1],player2Score[0]}};
                        4'b0011:{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{currentPlayer,4'b0,4'b0,4'b0,4'b0,4'b0,
{player3Score[7],player3Score[6],player3Score[5],player3Score[4]},
{player3Score[3],player3Score[2],player3Score[1],player3Score[0]}};
                        4'b0100:{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{currentPlayer,4'b0,4'b0,4'b0,4'b0,4'b0,
{player4Score[7],player4Score[6],player4Score[5],player4Score[4]},
{player4Score[3],player4Score[2],player4Score[1],player4Score[0]}};
                        default:{bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}=
{currentPlayer,4'b0,4'b0,4'b0,4'b0,4'b0,4'b0,4'b0};
                        endcase
                        end
          5'b10000:                          //计时状态
                  begin
```

```
                      {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}<=
{4'b0,4'b0,4'b0,4'b0,4'b0,4'b0,
{dataTimer[7],dataTimer[6],dataTimer[5],dataTimer[4]},
{dataTimer[3],dataTimer[2],dataTimer[1],dataTimer[0]}};
                      bit_control<=8'b00000011;
                  end
        default:    //报错状态
        begin
         bit_control<=8'b11111000;
        {bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0}={{1'b0,1'b0,1'b0,state[4]},
{1'b0,1'b0,1'b0,state[3]},{1'b0,1'b0,1'b0,state[2]},{1'b0,1'b0,1'b0,state[1]},
{1'b0,1'b0,1'b0,state[0]},4'd6,4'd7,4'd8};
         end
endcase
end
endmodule
```

scan the 8 segment

```
module scan_show(input clk,
                 input [3:0]bit7, `timescale 1ns / 1ps
module scan_show(input clk,
                 input [3:0]bit7,                    //七段数码管的八个不同位上的数
字
                 input [3:0]bit6,
                 input [3:0]bit5,
                 input [3:0]bit4,
                 input [3:0]bit3,
                 input [3:0]bit2,
                 input [3:0]bit1,
                 input [3:0]bit0,
                 input [7:0] encontrol,              //控制该位显示不显示，如果是
1，则显示
                 input [7:0] en_dp,                 //控制小数点是否显示
                 output reg [7:0]seg_en,
                 output [7:0]seg_out
);
reg tick=0;
reg [30:0]counter=0;
always@(posedge clk)                                //分频，为了仿真，频率已调高
begin
 if(counter==2)
 begin
 counter<=0;
 tick<=~tick;
 end
 else
 counter<=counter+1;
end

reg [3:0] segLocation=0;
always@(posedge tick)                                    //扫描
begin
if(segLocation==7)
segLocation<=0;
else
segLocation<=segLocation+1;
end

reg [3:0]presentNum=0;
numberToSegout numberToSeg1(presentNum,segLocation,en_dp,seg_out);

always@(posedge tick)
begin
    case(segLocation)

        0: if(encontrol[0]) begin seg_en<=8'b1111_1110; presentNum<=bit0; end
else seg_en<=8'b1111_1111;      //扫描显示，不同数码管的不同位
        1: if(encontrol[1]) begin seg_en<=8'b1111_1101; presentNum<=bit1; end
else seg_en<=8'b1111_1111;
        2: if(encontrol[2]) begin seg_en<=8'b1111_1011; presentNum<=bit2; end
else seg_en<=8'b1111_1111;
        3: if(encontrol[3]) begin seg_en<=8'b1111_0111; presentNum<=bit3; end
else seg_en<=8'b1111_1111;
        4: if(encontrol[4]) begin seg_en<=8'b1110_1111; presentNum<=bit4; end
else seg_en<=8'b1111_1111;
        5: if(encontrol[5]) begin seg_en<=8'b1101_1111; presentNum<=bit5; end
else seg_en<=8'b1111_1111;
        6: if(encontrol[6]) begin seg_en<=8'b1011_1111; presentNum<=bit6; end
else seg_en<=8'b1111_1111;
        7: if(encontrol[7]) begin seg_en<=8'b0111_1111; presentNum<=bit7; end
else seg_en<=8'b1111_1111;
    endcase
end

 endmodule
                 input [3:0]bit6,
                 input [3:0]bit5,
                 input [3:0]bit4,
                 input [3:0]bit3,
                 input [3:0]bit2,
                 input [3:0]bit1,
                 input [3:0]bit0,
                 input [7:0] encontrol,
                 input [7:0] en_dp,
                 output reg [7:0]seg_en,
                 output [7:0]seg_out
);
reg tick=0;
reg [30:0]counter=0;
always@(posedge clk)
begin
 if(counter==2)
//if(counter==2)
 begin
 counter<=0;
 tick<=~tick;
 end
 else
 counter<=counter+1;
end

reg [3:0] segLocation=0;
always@(posedge tick)
begin
```

```
if(segLocation==7)
segLocation<=0;
else
segLocation<=segLocation+1;
end

reg [3:0]presentNum=0;
numberToSegout numberToSeg1(presentNum,segLocation,en_dp,seg_out);

always@(posedge tick)
begin
    case(segLocation)
        0: if(encontrol[0]) begin seg_en<=8'b1111_1110; presentNum<=bit0; end
else seg_en<=8'b1111_1111;
        1: if(encontrol[1]) begin seg_en<=8'b1111_1101; presentNum<=bit1; end
else seg_en<=8'b1111_1111;
        2: if(encontrol[2]) begin seg_en<=8'b1111_1011; presentNum<=bit2; end
else seg_en<=8'b1111_1111;
        3: if(encontrol[3]) begin seg_en<=8'b1111_0111; presentNum<=bit3; end
else seg_en<=8'b1111_1111;
        4: if(encontrol[4]) begin seg_en<=8'b1110_1111; presentNum<=bit4; end
else seg_en<=8'b1111_1111;
        5: if(encontrol[5]) begin seg_en<=8'b1101_1111; presentNum<=bit5; end
else seg_en<=8'b1111_1111;
        6: if(encontrol[6]) begin seg_en<=8'b1011_1111; presentNum<=bit6; end
else seg_en<=8'b1111_1111;
        7: if(encontrol[7]) begin seg_en<=8'b0111_1111; presentNum<=bit7; end
else seg_en<=8'b1111_1111;
    endcase
end

 endmodule
```

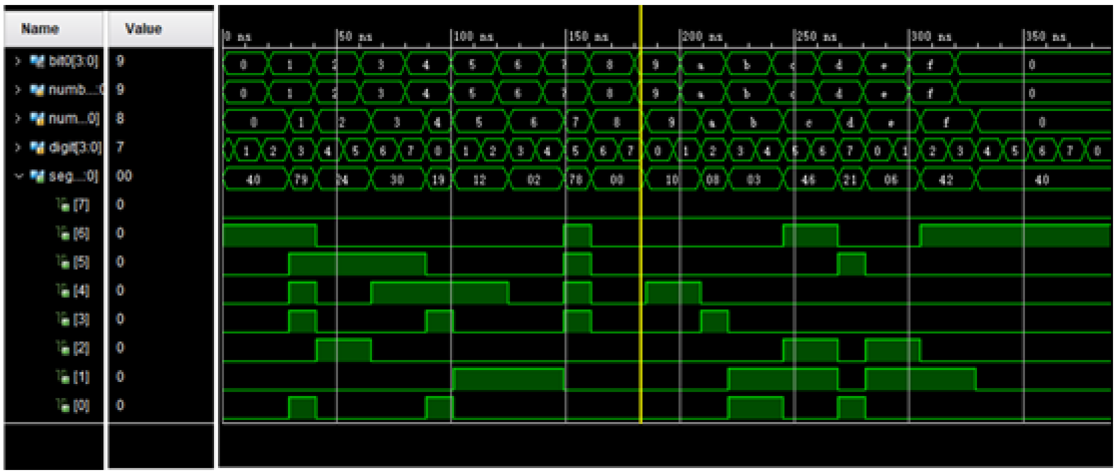
transverter**: transform the number to seg-out number.**

```
module numberToSegout(numDecimal, digit, en_p ,seg_out);
input [3:0] numDecimal;
input [3:0] digit;
input [7:0] en_p;
output [7:0] seg_out;

reg [6:0] seg_outNum;
reg en_point;

assign seg_out = {en_point,seg_outNum};

always @*
begin
    en_point = en_p[digit];
end

always @*
begin
    case(numDecimal)
        4'b0000: seg_outNum = 7'b1000000;
        4'b0001: seg_outNum = 7'b1111001;
        4'b0010: seg_outNum = 7'b0100100;
        4'b0011: seg_outNum = 7'b0110000;
        4'b0100: seg_outNum = 7'b0011001;
        4'b0101: seg_outNum = 7'b0010010;
        4'b0110: seg_outNum = 7'b0000010;
        4'b0111: seg_outNum = 7'b1111000;
        4'b1000:seg_outNum = 7'b0000000;
        4'b1001: seg_outNum= 7'b0010000;
        4'b1010: seg_outNum = 7'b0001000;
        4'b1011: seg_outNum = 7'b0000011;
        4'b1100: seg_outNum = 7'b1000110;
        4'b1101: seg_outNum= 7'b0100001;
        4'b1110: seg_outNum = 7'b0000110;
        4'b1111: seg_outNum= 7'b1000010; //为了显示GO，这一位显示的是G
        default: seg_outNum = 7'b1111111;
    endcase
end
endmodule
```

Simulation :

```
module show_tb();
wire clk;
reg clk1=0;
assign clk=clk1;
wire [3:0]bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0;
reg[3:0]number=4'b0000;
wire [7:0]seg_en;
wire [7:0]seg_out;
assign bit7=number;
assign bit6=number;
assign bit5=number;
assign bit4=number;
assign bit3=number;
assign bit2=number;
assign bit1=number;
assign bit0=number;
scan_show scan(clk,
                 bit7,
                 bit6,
                 bit5,
                 bit4,
                 bit3,
                 bit2,
                 bit1,
                 bit0,
                 8'b11111111,
                 8'b00000000,
                 seg_en,
                 seg_out
);
initial
forever
#1 clk1=~clk1;
initial
begin
repeat(16)
```

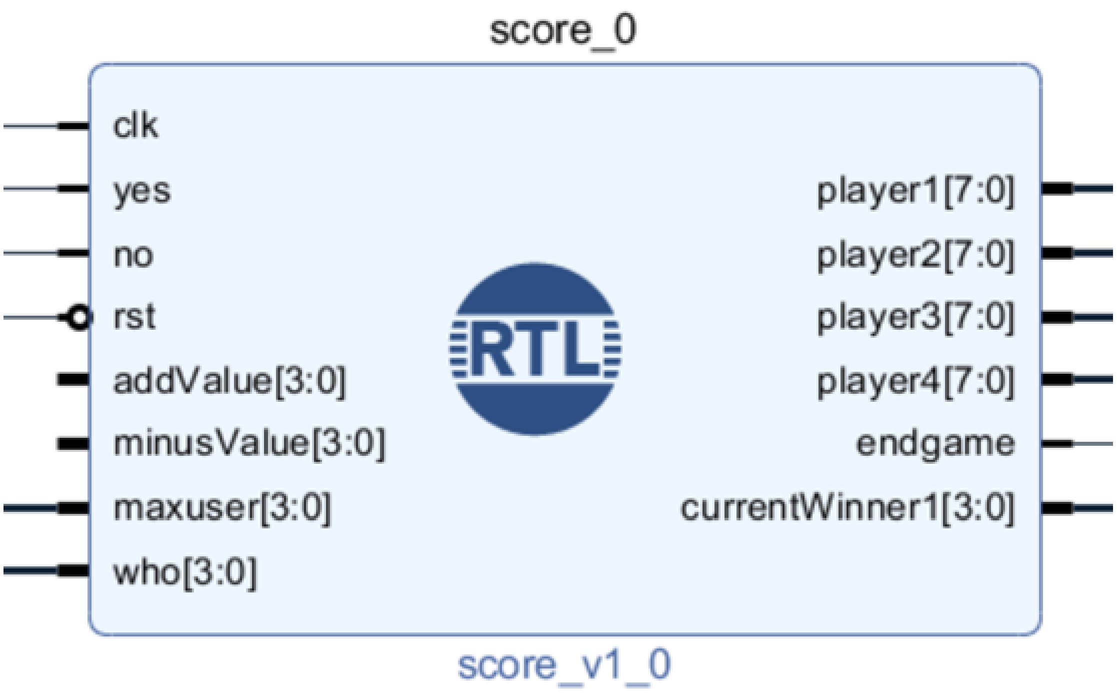
```
#20 number=number+1;
end
endmodule
```



As the simulation shows, with the seg_out can represent correct number of the bit.

6. Score_part

Port description



Input:

Clk: the system clock;

Yes: add score to the player

No:minus score to the player

Rst: reset the score to 0

addValue\minusValue\maxUser: read the data of setting part

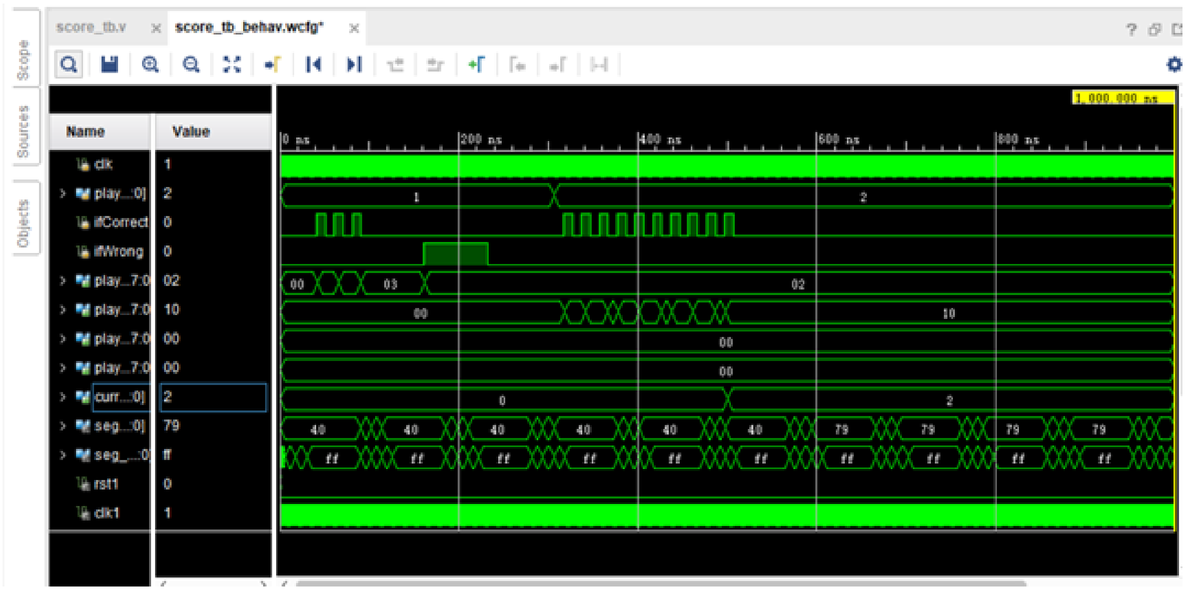
who:current player who can answer.

Output:

Player1\2\3\4:the curentscore

Endgame: if one player reacher 10 score,the endgame will be 1.

currentWinner1:the player who get 10 scores first.



As we can see, the player1 score add to 3,and minus to 2.Then we switch the current player to player2 and let him get 10 scores. Then the “endgame will be 1”, and “currentWinner1” changed from 0 to 2.

Design idea

```
module score(
            input
            clk,
            yes,
            no,
            rst,
            input [3:0]scorejia,
            [3:0]scorejian,
            input [3:0]maxuser,
            [3:0]who,
            output wire[7:0] player1,
            wire[7:0] player2,
            wire[7:0] player3,
            wire[7:0] player4,
            wire endgame,
            output  [3:0]currentWinner1
    );
    reg[7:0] player11;reg[7:0] player21;reg[7:0] player31;reg[7:0] player41;
    assign player1=(player11/10)*16+player11%10;
    assign player2=(player21/10)*16+player21%10;
    assign player3=(player31/10)*16+player31%10;
    assign player4=(player41/10)*16+player41%10;
   assign endgame=(player1>=10 )||(player2>=10)||(player3>=10) ||(player4>=10);
    reg [3:0]currentWinner=4'b0;
    assign currentWinner1=currentWinner;
    reg [31:0]counter=0;
    reg clk_out=0;
    always@(posedge clk,posedge rst)                //分频器
```

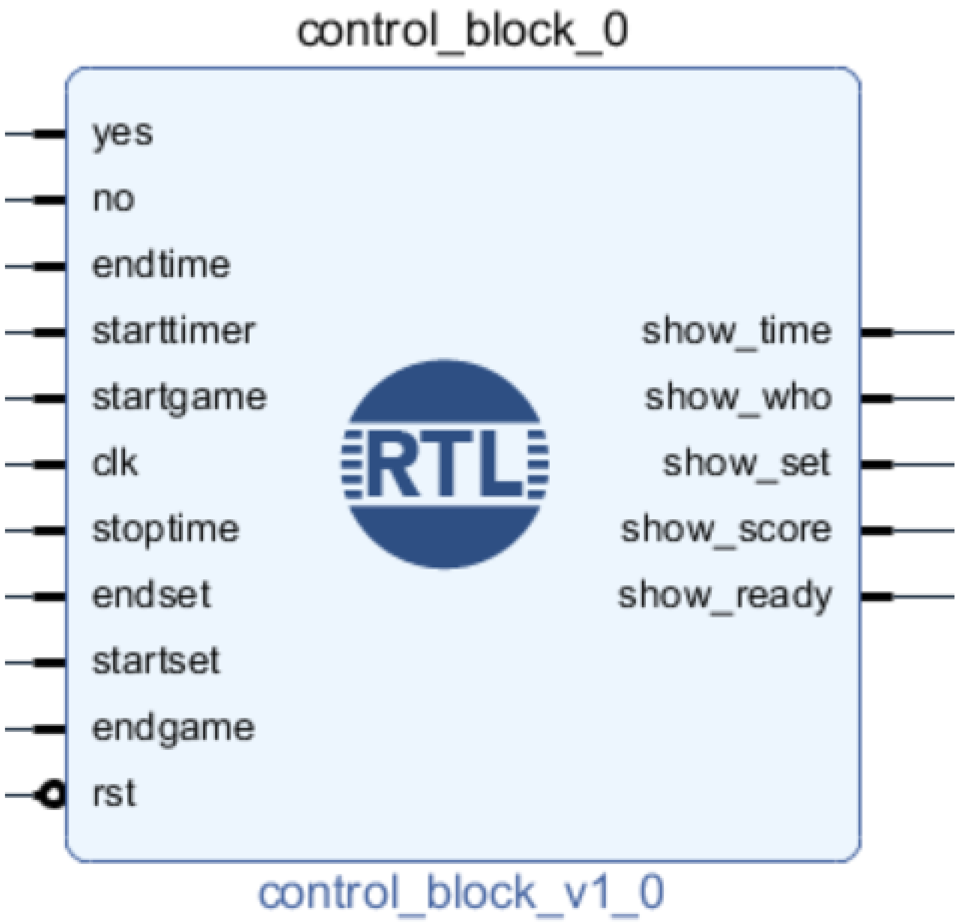
```
     if(rst)
     begin
     counter<=0;
     clk_out<=1'b0;
     end
     else begin
         if(counter==2)
             begin
             counter<=0;
             clk_out<=~clk_out;
              end
          else
         counter<=counter+1;
     end
     always@(posedge clk)
     begin
     if(endgame)                                                  //judge the winner
     begin
     if(player1>=10) currentWinner<=4'b0001;
     else if(player2>=10) currentWinner<=4'b0010;
     else if (player3>=10) currentWinner<=4'b0011;
     else if(player4>=10) currentWinner<=4'b0100;
     else currentWinner<=4'b0000;//if error,it occurs
     end
     end
     reg [3:0]currentplayer=4'b0000;
     reg [3:0]nextplayer=4'b0000;
     reg [7:0]nextplayer1=4'b0000;
     reg [7:0]nextplayer2=4'b0000;
     reg [7:0]nextplayer3=4'b0000;
     reg [7:0]nextplayer4=4'b0000;
     //  reg [3:0]winnernext;
     // reg nextendgame=0;
     always@(posedge clk_out,posedge rst)
     if(rst)
     begin
     currentplayer<=4'b0000;
     player11<=0;
     player21<=0;
     player31<=0;
     player41<=0;
     //endgame<=0;
     end
     else
     begin
     currentplayer<=nextplayer;
     player11<=nextplayer1;
     player21<=nextplayer2;

     player31<=nextplayer3;
     player41<=nextplayer4;
     //  endgame<=nextendgame;
     end
     always@(posedge yes,posedge no)
     begin
     if(yes)
    begin
     case (who)
    4'b0001: nextplayer1=player11+scorejia;

        //add score
    4'b0010:nextplayer2=player21+scorejia;
    4'b0011: nextplayer3=player31+scorejia;
    4'b0100: nextplayer4=player41+scorejia;
    endcase
    end
   else  if(no)
    case (who)
        4'b0001:nextplayer1=(player11>scorejian)?player11-scorejian:4'b0;
        //reduce score
        4'b0010: nextplayer2=(player21>scorejian)?player21-scorejian:4'b0;
        4'b0011: nextplayer3=(player31>scorejian)?player31-scorejian:4'b0;
        4'b0100: nextplayer4=(player41>scorejian)?player41-scorejian:4'b0;
        endcase
    else
    begin
    nextplayer1=player11;
 //有限状态机
    nextplayer2=player21;
    nextplayer3=player31;
    nextplayer4=player41;
    end
    end
    endmodule
```

Simulation

```
module show_tb();
wire clk;
reg clk1=0;
assign clk=clk1;
wire [3:0]bit7,bit6,bit5,bit4,bit3,bit2,bit1,bit0;
reg[3:0]number=4'b0000;
wire [7:0]seg_en;
wire [7:0]seg_out;
assign bit7=number;
assign bit6=number;
assign bit5=number;
assign bit4=number;
assign bit3=number;
assign bit2=number;
assign bit1=number;
assign bit0=number;
scan_show scan(clk,
                 bit7,
                 bit6,
                 bit5,
                 bit4,
                 bit3,
                 bit2,
                 bit1,
                 bit0,
                 8'b11111111,
                 8'b00000000,
                 seg_en,
                  seg_out
);
initial
forever
#1 clk1=~clk1;
initial
begin
repeat(16)
#20 number=number+1;
end
endmodule
`timescale 1ns / 1ps
module test_Score(input rst,
                   input clk,
                   input enable,
                   input [3:0]player,
                   input ifCorrect,
                   input ifWrong,
                   output [7:0]seg_out,
                   output [7:0]seg_en,
                   output [3:0]num,
                   output [3:0]test222
                   );
                   assign test222=player;
                   wire [31:0]score_out;
                   wire [3:0]testplayer;
                   assign testplayer=player;
                   wire[7:0] player1score,
                   player2score,
                   player3score,
                   player4score;
                   wire endgame;
                   wire [3:0]winner;
    assign num=score_out;
    seg7 show(rst,
               clk,
                5'b01000,//state,
                0,      //dataTimer
                //input [7:0] dataChange,
                testplayer,//input [3:0] dataWho,
                testplayer,//input [3:0] currentPlayer,
                player1score,
                player2score,
                player3score,
                player4score,
                0,
                0,
                0,
                0,//input [31:0] playersScore,
                0,// input [7:0] settings,
                winner,//input [7:0] controller,
                seg_out,//output [7:0]seg_out,
                seg_en//output [7:0]seg_en,
                );
                score scorePart(
                    clk,
                    ifCorrect,
                    ifWrong,
                    rst,
                    4'b0001,
                    4'b0001,
                    4'b0100,
                    player,
                    player1score,
                    player2score,
                    player3score,
                    player4score,
                    endgame,
                    winner
                    );
endmodule
```

7. Control

Port description



Input:

Yes\no:read from button

Endtime:read from scorepart

Start timer: begin the timer

Startgame: begin a new turn

Clk: system clock

Stoptime: end the timer

Endset end the setting

Startset begin the setting

Endgame:if the winner exists, it will be 1

Rst:reset the system and begin a new game

Design idea:

```
module control_block(
    input
yes,no,endtime,starttimer,startgame,clk,stoptime,endset,startset,endgame,rst,
    output  show_time,
    output  show_who,
    output  show_set,
    output  show_score,
    output  show_ready
    );
     reg [4:0]statereg=5'b00001;
    assign show_time=statereg[4];
    assign show_who=statereg[3];
    assign show_set=statereg[2];
    assign show_score=statereg[1];
    assign show_ready=statereg[0];

    reg [4:0]statenext;
    always@(posedge clk,posedge rst)
    if(rst)
    statereg<=5'b00001;
    else
    statereg<=statenext;
always @*
case(statereg)
5'b00001:if(starttimer)statenext=5'b10000;else
if(startset)statenext=5'b00100;else if(endgame) statenext=5'b00010;else
statenext=5'b00001;//go
5'b10000: if(stoptime)statenext=5'b01000; else
if(endtime)statenext=5'b00001;else statenext=5'b10000; //计时
5'b01000:if(startgame)statenext=5'b00001;else statenext=5'b01000;
                    //计分
5'b00100:if(endset)statenext=5'b00001;else statenext=5'b00100;
                  //设置
5'b00010:statenext=5'b00010;
                   //结束
endcase
endmodule
```

Constraints

code:

```
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en_0[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out_0[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports bee]
set_property PACKAGE_PIN A18 [get_ports {seg_en_0[7]}]
set_property PACKAGE_PIN A20 [get_ports {seg_en_0[6]}]
set_property PACKAGE_PIN B20 [get_ports {seg_en_0[5]}]
set_property PACKAGE_PIN E18 [get_ports {seg_en_0[4]}]
set_property PACKAGE_PIN F18 [get_ports {seg_en_0[3]}]
set_property PACKAGE_PIN D19 [get_ports {seg_en_0[2]}]
set_property PACKAGE_PIN E19 [get_ports {seg_en_0[1]}]
set_property PACKAGE_PIN C19 [get_ports {seg_en_0[0]}]
set_property PACKAGE_PIN E13 [get_ports {seg_out_0[7]}]
set_property PACKAGE_PIN C15 [get_ports {seg_out_0[6]}]
set_property PACKAGE_PIN C14 [get_ports {seg_out_0[5]}]
set_property PACKAGE_PIN E17 [get_ports {seg_out_0[4]}]
set_property PACKAGE_PIN F16 [get_ports {seg_out_0[3]}]
set_property PACKAGE_PIN F14 [get_ports {seg_out_0[2]}]
set_property PACKAGE_PIN F13 [get_ports {seg_out_0[1]}]
set_property PACKAGE_PIN F15 [get_ports {seg_out_0[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports player1]
set_property IOSTANDARD LVCMOS33 [get_ports player2]
set_property IOSTANDARD LVCMOS33 [get_ports player3]
set_property IOSTANDARD LVCMOS33 [get_ports player4]
set_property IOSTANDARD LVCMOS33 [get_ports starttimer]
set_property IOSTANDARD LVCMOS33 [get_ports no]
set_property IOSTANDARD LVCMOS33 [get_ports yes]
set_property IOSTANDARD LVCMOS33 [get_ports set1]
set_property PACKAGE_PIN U5 [get_ports player1]     // 选手1到选手4的抢答
set_property PACKAGE_PIN T5 [get_ports player2]
set_property PACKAGE_PIN T4 [get_ports player3]
set_property PACKAGE_PIN R4 [get_ports player4]
set_property PACKAGE_PIN V9 [get_ports starttimer]   // 主持人计时
set_property PACKAGE_PIN R1 [get_ports no]           // 主持人判定错误
set_property PACKAGE_PIN P1 [get_ports yes]         // 主持人判定正确
set_property PACKAGE_PIN R6 [get_ports set1]      //主持人设置
set_property PACKAGE_PIN Y18 [get_ports clk]
set_property PACKAGE_PIN Y9 [get_ports rst]    //重新开始比赛
set_property IOSTANDARD LVCMOS33 [get_ports rst]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports startgame]
set_property PACKAGE_PIN W4 [get_ports startgame]
set_property PACKAGE_PIN A19 [get_ports bee]
set_property IOSTANDARD LVCMOS33 [get_ports jia1]
set_property IOSTANDARD LVCMOS33 [get_ports jia2]
set_property IOSTANDARD LVCMOS33 [get_ports jian1]
set_property IOSTANDARD LVCMOS33 [get_ports jian2]
set_property IOSTANDARD LVCMOS33 [get_ports time1]
set_property IOSTANDARD LVCMOS33 [get_ports time2]
set_property PACKAGE_PIN W9 [get_ports jia1]     //设置比赛加分值、减分值、计时时长
set_property PACKAGE_PIN Y7 [get_ports jia2]
set_property PACKAGE_PIN Y8 [get_ports jian1]
set_property PACKAGE_PIN AB8 [get_ports jian2]
set_property PACKAGE_PIN AA8 [get_ports time1]
set_property PACKAGE_PIN V8 [get_ports time2]
set_property SEVERITY {Warning} [get_drc_checks UCIO-1]
```

Problems and Solution:

Problem 1.

The posedge trigger signal can work in the simulation, but when we program it in the minisys board, the signal can't trigger the system. (like the signal judge the answer is correct or wrong)

Solution 1:

This problem is because the “always” block can't not implement the real circuit, even if the simulation is all right. Therefore, we need to use finite-state machine.

Let the signal can be triggered in a certain state, and the signal does not need to be posedge, but just Synchronize with the clock.

Problem 2

The score may add to a wrong player, or the sore value is larger than it should add.

Solution 2:

The reason is that our frequency dividers do not conform with each other. The score parts' clock signal is too fast, so it may read multiple “add” signal even if we just push the button only once. Therefore, we adjust the frequency part, and the problem is solved.

Summary

Some Improvements After the Show in Class

We try to realize the unfinished review function to check the score of each player in this competition.

We have successfully finished the design and simulation part.

The modelu Retrieve(This module is unfinished and we will accomplish it in future)

Design idea:

Array is used to implement the storage function, and the game state of each round is stored after the end of each round. After the end of the round, the results of each round can be retrieved according to the information stored in the array.

The signal switch==1 means write data, while switch==0 means read data.

When you write data in the array, you can replace the clk signal with another signal which represents the accomplishment of one round of the competition.

When you read data in the array, You can manually control the loop to read the elements in the array.

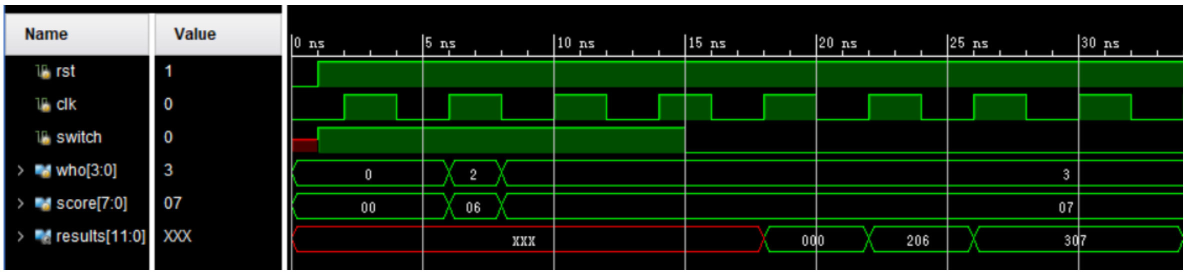
But in the original design, data changes if driven by the posedge of the clock.

Design code:

```
module Retrieve(
input rst,clk,[3:0]who,[7:0]score,
 switch,//when switch==1,it means writing data while when switch==0,it means
reading data
output reg[11:0] results
);
reg [5:0]count;
reg [11:0] data[1:20];
reg [10:0]index=1;
always@(posedge clk,negedge rst) begin
    if(~rst)begin//异步复位信号rst实现循环计数
    count=1;
    index=1;
end
else begin
    if(switch)begin
    data[count]={who,score};
    count=count+1;
end
else begin
    if(index<=count)begin
    results=data[index];
    index=index+1;
    end
end
end
end
endmodule
```

Simulation:

```
module Retrieve_sim();
reg rst,clk,switch;
reg [3:0]who;
reg [7:0]score;
wire [11:0] results;
Retrieve u1(rst,clk,who,score,switch,results);
always#2clk=~clk;
initial fork
    clk=0;
    rst=0;
    score=8'b00000000;
    who=4'b0000;
    #1rst=1;
    #1switch=1;
    #6who=4'b0010;
    #6score=8'b00000110;
    #8who=4'b0011;
    #8score=8'b00000111;
    #15switch=0;
join
endmodule
```



We can see that when switch=1, it is the stage of writing data and the data of three rounds have been put into the array. When switch=0, it is the stage of reading data and the results successfully display its elements cyclically. Note that all the performance is driven by the posedge of the clk. The results of simulation meets our expectations.

Main Features of the Current System

1. Successfully implemented the basic functions of the answering device. After the answering time, each person's score and the final winner can be displayed in a loop.
2. Successfully implemented the setting function, which can set the number of participants and the countdown time of the responder and the score of each question
3. Successfully implemented the buzzer function with different music, which can play different music as a reminder at different stages of the answer

Optimization Direction

1. Realize the unfinished review function to check the score of each player in this competition. In the previous design and simulation of the module Retrieve, data was driven by the posedge of the clock. In practice, When you write data in the array, you can replace the clk signal with another signal which represents the accomplishment of one round of the competition. When you read data in the array, You can manually control the loop to read the elements in the array.
2. We will implement some interesting additional functions in the answering device, such as adding keypad input to control the answering process, improving the stability and practicality of the system.

Verification on Mini-sys

We have already shown our actual effect in front of classmates and teachers in the classroom, so the demonstration video is omitted here.