上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

FPGA experiment

THESIS OF CURRICULUM

Rhythm Master

An Interactive Game on BASYS2 Platform

NAME: Weiming Bao

STUDENT I.D.: 5140219191

MAJOR: Computer Science

INSTRUCTOR: Li, Dan

SCHOOL: S.E.I.E.E.

Contents

Rhythm Master

—— An Interactive Game on BASYS2

1 Aim

This project is aimed to design an interactive game called ***"Rhythm Master"***, which is inspired by a hot smartphone game of the same name on Basys2 platform, utilizing the buttons, switches and 7-segment LEDs on board via Xilinx IDE.

(**Pic1.** LOGO of "Rhythm Master")

2 How to play

The DEMO video is uploaded online. (URL: https://s3-ap-northeast-1.amazonaws.com/wmbao-fpga/5140219191_RhythmMaster_DEMO.wmv)

- After the player turns on the "power" switch, the game begins. The beep starts to play the background music and arbitrary digit on board is going to become active, which means it'll be lit one-segment-by-one-segment, as is shown *in Pic2* below.

(**Pic2.** the sequence of segments to be lit when active)

- The player should press the corresponding button below exactly when the 7th segment, which is the last one of a digit, turns on. Each time the player make it, he/she will gain 1 point, while the player will lose 1 point once he/she mis-push the button or at the wrong time.

- When the music ends, the score is displayed on the 4 digits LEDs, in form of "S(-)XX", as is illustrated *in Pic3 and Pic4* below. The game is actually a little tough, negative score is also possible.

(**Pic3.** Score: +08)

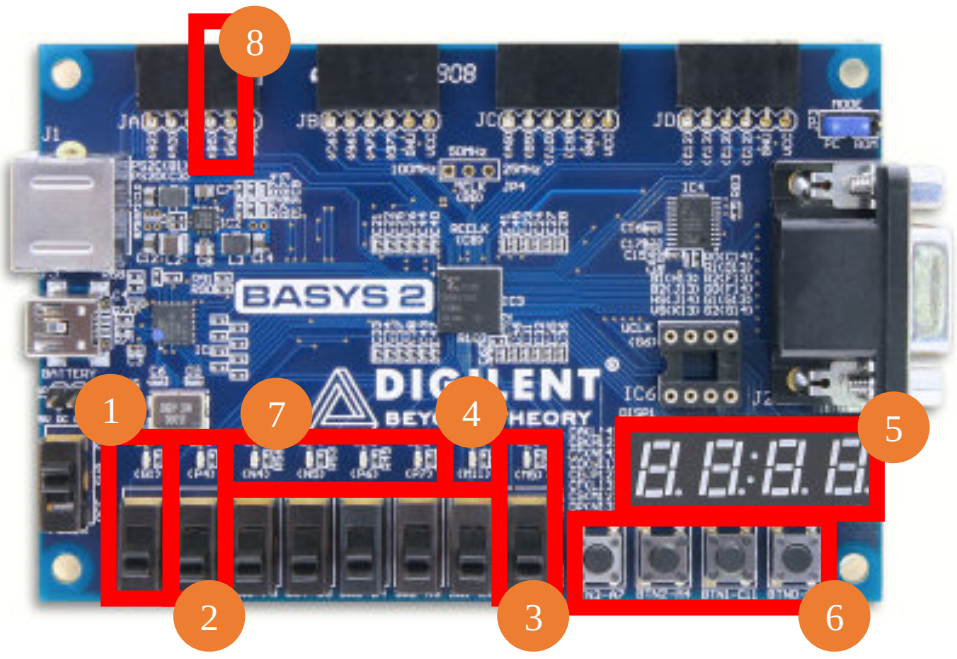
(**Pic4.** Score: -07)

- During the game, the player can pause or reset whenever he wants to.

3 Design

3.1 Hardware Interface

The following picture *(Pic5)* shows all the components on board used in this project with their function briefly explained below.



(**Pic5.** the components used on board)

① "on/off" switch

② "pause" switch

③ "reset" switch

④ “ending” (signal)

⑤ 4 7-segment digital display

⑥ 4 buttons (corresponding to for digits on board)

⑦ 4 LEDs (signals showing whether the corresponding button is pushed)

⑧ beep pin (B5) –connected to the additional hardware a non-source beep (*Pic6).*

(**Pic6.** the non-source beep – SFN3040 - used for background music)

3.2 Software Structure

- I have divided the whole design into six modules.

  ① the Control Module (CTR),

  ② the Frequency Divider Module (fre_div),

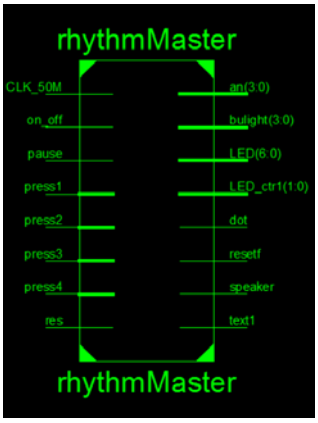
  ③ the Digits Display Module (LED_display),

  ④ the Beep Drive Module (beep),

  ⑤ the Segment Sub Module for digits (seg_dig_sub),

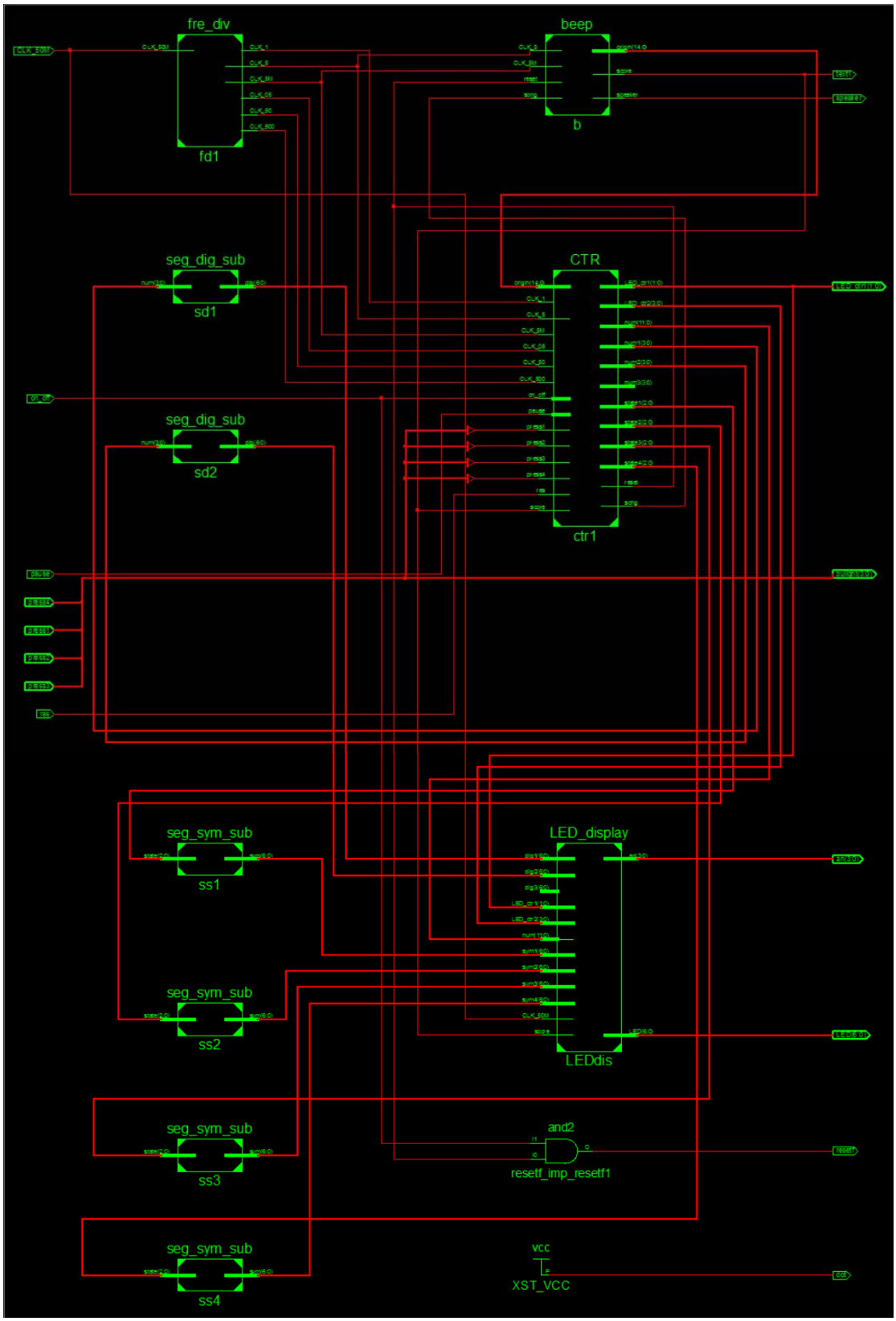
  ⑥ the Segment Sub Module for symbols (seg_sym_sub).

- The top-level schematic of the whole program is shown *in Pic7* below.



(**Pic7.** top-level schematic of *"Rhythm Master"*)

- The RTL schematic is shown *in Pic8* below.



(**Pic8.** the RTL schematic for *"Rhythm Master"*)

- The whole source code is attached in the Appendix.

4 Reflection

The whole project is a little complicated. I separated it into 6 modules and used wires and buses to connect those modules. During the programing, I met quite a lot of problems. Luckily, I have got many help, solving most of the problem, from the online forum run by Digilent company, which is well-organized and quite helpful. I list three main problems I've met with below along with my solutions and related thoughts.

- **Problem1:** Limited number of variables supported in sensitivity list

  Solution1: Use a flag register to hold the result of the logic composed by multiple variables and put the flag register into the sensitivity list.

  Solution2: Check the program carefully, and it's possible there're some collision.

- **Problem2:** Separate the whole design into modules and cooperate

  Solution : Before we program, we need to make it clear what we'd like to build and have the whole draft of the structure of functions and modules in mind.

- **Problem3:** Non-source beep driver (generate frequency manually)

  Solution : The non-source beep is not integrated on board, and there's little reference of using non-source beep available online. My solution is manually generate the frequency of the specific melody, inspired by PWM, the source code of which is attached in 5.3

5 Appendix

5.1 CTR.v

```
module CTR(
    input on_off,
```

```
        input pause,
        input res,
        input CLK_5M,
        input CLK_500,
        input CLK_50,
        input CLK_5,
        input CLK_1,
        input CLK_05,
        input score,
        input press1,
        input press2,
        input press3,
        input press4,
        input [14:0] origin,
        output reg song,
        output reg reset,
        output wire [1:0] LED_ctr1,
        output reg [3:0] LED_ctr2,
        output reg [2:0] state1,
        output reg [2:0] state2,
        output reg [2:0] state3,
        output reg [2:0] state4,
        output wire [3:0] num1,
        output wire [3:0] num2,
        output wire [3:0] num3,
        output reg [11:0] num
        );

reg [3:0] cnt_10;

wire flag;
assign flag = press1 + press2 + press3 + press4;
assign num3 = num[11:8];
assign num2 = num[7:4];
assign num1 = num[3:0];

initial begin
    state1 <= 3'b000;
    state2 <= 3'b000;
    state3 <= 3'b000;
    state4 <= 3'b000;
    LED_ctr2 <= 4'b0000;
    num <= 12'b0;
    cnt_10 <= 4'b0000;
end

assign LED_ctr1[0] = on_off;
assign LED_ctr1[1] = pause;

always @(posedge CLK_5) begin
```

```
    LED_ctr2 = 4'b0000;
    if (song && (!score)) begin
        cnt_10 = cnt_10 + 1'b1;
        if (cnt_10 == 4'd9) begin
            cnt_10 = 4'd0;
            if ((origin[11] == 1)&&(state1 == 3'b000)) begin
                state1 = 3'b001;
            end
            if ((origin[7] == 1)&&(state2 == 3'b000)) begin
                state2 = 3'b001;
            end
            if ((origin[5] == 1)&&(state3 == 3'b000)) begin
                state3 = 3'b001;
            end
            if ((origin[1] == 1)&&(state4 == 3'b000)) begin
                state4 = 3'b001;
            end
        end

        if (state1 > 3'b000)
            state1 = state1 + 1'b1;
        if (state2 > 3'b000)
            state2 = state2 + 1'b1;
        if (state3 > 3'b000)
            state3 = state3 + 1'b1;
        if (state4 > 3'b000)
            state4 = state4 + 1'b1;
    end
    else begin
        if (score) begin
            LED_ctr2 = 4'b0000;
        end
        if (reset) begin
            LED_ctr2 = 4'b1111;
        end
        state1 = 3'b000;
        state2 = 3'b000;
        state3 = 3'b000;
        state4 = 3'b000;
    end
end

always @(*) begin
    if (on_off == 1) begin
        if (res == 0) begin
            reset <= 0;
            if (pause == 0) begin
                song <= 1;
            end
            else song <= 0;
```

```
            end
            else begin
                reset <= 1;
                song <= 0;
            end
        end
        else begin
            song <= 0;
            reset <= 1;
        end
    end

    always @(posedge flag)
        begin
            if (song && (!score)) begin
                if (press1) begin
                    if (state1 == 3'b111)
                        num = num + 1;
                    else begin
                        //if (num > 1'd0)
                            num = num - 1;
                    end
                end
                else if (press2) begin
                    if (state2 == 3'b111)
                        num = num + 1;
                    else begin
                        //if (num > 1'd0)
                            num = num - 1;
                    end
                end
                else if (press3) begin
                    if (state3 == 3'b111)
                        num = num + 1;
                    else begin
                        //if (num > 1'd0)
                            num = num - 1;
                    end
                end
                else if (press4) begin
                    if (state4 == 3'b111)
                        num = num + 1;
                    else begin
                        //if (num > 1'd0)
                            num = num - 1;
                    end
                end
            end
            else if (reset) begin
                num = 12'b0;
```

```
            end
        end

    endmodule
```

5.2 fre_div.v

```
module fre_div(
        input CLK_50M,
        output reg CLK_5M,
        output reg CLK_500,
        output reg CLK_50,
        output reg CLK_5,
        output reg CLK_1,
        output reg CLK_05
        );

reg[3:0] cnt1; //频率 5MHz
reg[16:0] cnt2; //频率 500Hz
reg[19:0] cnt3;//频率 50Hz
reg[23:0] cnt4;//频率 5Hz
reg[25:0] cnt5;//频率 1Hz
reg[27:0] cnt6;//频率 0.5Hz

always @(posedge CLK_50M)
    begin
        cnt1<=cnt1+1'b1;
         cnt2<=cnt2+1'b1;
         cnt3<=cnt3+1'b1;
        cnt4<=cnt4+1'b1;
         cnt5<=cnt5+1'b1;
         cnt6<=cnt6+1'b1;
        if(cnt1==4'd9)
            begin
                cnt1<=4'd0;
                CLK_5M <= ~CLK_5M;
            end
         if(cnt2==17'h1869F)
            begin
                cnt2<=17'h0;
                CLK_500 <= ~CLK_500;
            end
         if(cnt3==20'hF423F)
            begin
                cnt3<=20'h0;
                CLK_50 <= ~CLK_50;
            end
        if(cnt4==24'h98967F)
```

```
                begin
                    cnt4<=24'h0;
                    CLK_5 <= ~CLK_5;
                end
            if(cnt5==26'h2FAF07F)
                begin
                    cnt5<=26'h0;
                    CLK_1 <= ~CLK_1;
                end
            if(cnt6==28'h5F5E0FF)
                begin
                    cnt6<=28'h0;
                    CLK_05 <= ~CLK_05;
                end
        end

    endmodule
```

5.3 beep.v

```
module beep(
        input CLK_5M,
        input CLK_5,
        input song,
        input reset,
        output reg speaker,
        output reg score,
        output reg [14:0] origin
        );

parameter wide=15;
reg[7:0] cnt;    //音名数
reg[wide-1:0] drive;
reg[1:0] count;
reg carrier;

initial begin
    cnt <= 140;
    score <= 0;
    end

always @(posedge CLK_5M)
    begin
        if (song && (!score)) begin
            if(drive==15'h7fff) begin
                drive<=origin;
                carrier<=1'b1;
                end
            else begin
```

```
                drive<=drive+1'b1;
                carrier<=1'b0;
            end
        end
    end

//carrier 的频率是每个音阶的频率

always @(posedge carrier)
    begin
        if (song && (!score)) begin
            count<=count+1'b1;
            if(count==4'd0)
                speaker<=1'b1;
            else speaker<=1'b0;
        end
    end

always @(posedge CLK_5 or posedge reset)
    begin
        if (reset == 1) begin
            cnt <= 0;
            score <= 0;
        end
        else if (song == 1) begin
            if (cnt<8'd139)
                cnt<=cnt+1'b1;
            else if(cnt == 8'd139)
                score <= 1;
            case (cnt)
                8'd0:origin<=15'h625F;   //中音 3，4 个节拍
                8'd1:origin<=15'h625F;
                8'd2:origin<=15'h625F;
                8'd3:origin<=15'h625F;
                8'd4:origin<=15'h6715; //中音 5,3 个节拍
                8'd5:origin<=15'h6715;
                8'd6:origin<=15'h6715;
                8'd7:origin<=15'h69cd;//中音 6
                8'd8:origin<=15'h6d55; //高音 1，3 个节拍
                8'd9:origin<=15'h6d55;
                8'd10:origin<=15'h6d55;

                8'd11:origin<=15'h6f5f; //高音 2
                8'd12:origin<=15'h69cd; //中音 6
                8'd13:origin<=15'h6d55; //高音 1
                8'd14:origin<=15'h6715;   //中音 5
                8'd15:origin<=15'h6715;
                8'd16:origin<=15'h738a; //高音 5
```

```
8'd17:origin<=15'h738a;
8'd18:origin<=15'h738a;
8'd19:origin<=15'h76aa; //倍高音 1
8'd20:origin<=15'h69cd; //高音 6

8'd21:origin<=15'h6715;//高音 5
8'd22:origin<=15'h712f;//高音 3
8'd23:origin<=15'h6715;//高音 5
8'd24:origin<=15'h6f5f; //高音 2
8'd25:origin<=15'h6f5f;
8'd26:origin<=15'h6f5f;
8'd27:origin<=15'h6f5f;
8'd28:origin<=15'h6f5f;
8'd29:origin<=15'h6f5f;
8'd30:origin<=15'h6f5f;
8'd31:origin<=15'h6f5f;
8'd32:origin<=15'h6f5f;
8'd33:origin<=15'h6f5f;
8'd34:origin<=15'h6f5f;
8'd35:origin<=15'h712f;//高音 3
8'd36:origin<=15'h6c39; //中音 7
8'd37:origin<=15'h6c39;
8'd38:origin<=15'h69cd;//中音 6
8'd39:origin<=15'h69cd;
8'd40:origin<=15'h6715; //中音 5
8'd41:origin<=15'h6715;
8'd42:origin<=15'h6715;
8'd43:origin<=15'h69cd;//中音 6
8'd44:origin<=15'h6d55;//高音 1
8'd45:origin<=15'h6d55;
8'd46:origin<=15'h6f5f;//高音 2
8'd47:origin<=15'h6f5f;
8'd48:origin<=15'h625f;//中音 3
8'd49:origin<=15'h625f;
8'd50:origin<=15'h6d55; //高音 1
8'd51:origin<=15'h6d55;

8'd52:origin<=15'h69cd;//中音 6
8'd53:origin<=15'h6715;//中音 5
8'd54:origin<=15'h69cd; //中音 6
8'd55:origin<=15'h6d55;//高音 1
8'd56:origin<=15'h6715;//中音 5
8'd57:origin<=15'h6715;
8'd58:origin<=15'h6715;
8'd59:origin<=15'h6715;
8'd60:origin<=15'h6715;
```

```
8'd61:origin<=15'h6715;
8'd62:origin<=15'h6715;
8'd63:origin<=15'h6715;
8'd64:origin<=15'h712f;//高音 3
8'd65:origin<=15'h712f;
8'd66:origin<=15'h712f;
8'd67:origin<=15'h738a;//高音 5
8'd68:origin<=15'h6c39;//中音 7
8'd69:origin<=15'h6c39;
8'd70:origin<=15'h6f5f;//高音 2
8'd71:origin<=15'h6f5f;

8'd72:origin<=15'h69cd; //中音 6
8'd73:origin<=15'h6d55;//高音 1
8'd74:origin<=15'h6715;//中音 5
8'd75:origin<=15'h6715;
8'd76:origin<=15'h6715;
8'd77:origin<=15'h6715;
8'd78:origin<=15'h6715;
8'd79:origin<=15'h6715;
8'd80:origin<=15'h625f; //中音 3

8'd81:origin<=15'h6715;//中音 5
8'd82:origin<=15'h625f;//中音 3
8'd83:origin<=15'h625f;
8'd84:origin<=15'h6715;//中音 5
8'd85:origin<=15'h69cd;//中音 6
8'd86:origin<=15'h6c39;//中音 7
8'd87:origin<=15'h6f5f;//高音 2
8'd88:origin<=15'h69cd;//中音 6
8'd89:origin<=15'h69cd;
8'd90:origin<=15'h69cd;
8'd91:origin<=15'h69cd;
8'd92:origin<=15'h69cd;
8'd93:origin<=15'h69cd;
8'd94:origin<=15'h6715;//中音 5
8'd95:origin<=15'h69cd;//中音 6
8'd96:origin<=15'h6d55;//高音 1
8'd97:origin<=15'h6d55;
8'd98:origin<=15'h6d55;
8'd99:origin<=15'h6f5f;////高音 2
8'd100:origin<=15'h738a; //高音 5
8'd101:origin<=15'h738a;
8'd102:origin<=15'h738a;
8'd103:origin<=15'h712f;//高音 3
8'd104:origin<=15'h6f5f;//高音 2
```

```
                8'd105:origin<=15'h6f5f;
                8'd106:origin<=15'h712f;//高音 3
                8'd107:origin<=15'h6f5f;//高音 2
                8'd108:origin<=15'h6d55;//高音 1
                8'd109:origin<=15'h6d55;
                8'd110:origin<=15'h69cd;//中音 6

                8'd111:origin<=15'h6715;//中音 5
                8'd112:origin<=15'h625f;//中音 3
                8'd113:origin<=15'h625f;
                8'd114:origin<=15'h625f;
                8'd115:origin<=15'h625f;
                8'd116:origin<=15'h6d55;//高音 1
                8'd117:origin<=15'h6d55;
                8'd118:origin<=15'h69cd;//中音 6
                8'd119:origin<=15'h6d55;//高音 1
                8'd120:origin<=15'h69cd;//中音 6

                8'd121:origin<=15'h625f;//中音 3
                8'd122:origin<=15'h625f;
                8'd123:origin<=15'h6f5f;//高音 2
                8'd124:origin<=15'h625f;//中音 3
                8'd125:origin<=15'h6715;//中音 5
                8'd126:origin<=15'h69cd;//中音 6
                8'd127:origin<=15'h6d55;//高音 1
                8'd128:origin<=15'h6715;//中音 5
                8'd129:origin<=15'h6715;
                8'd130:origin<=15'h6715;
                8'd131:origin<=15'h6715;
                8'd132:origin<=15'h6715;
                8'd133:origin<=15'h6715;
                8'd134:origin<=15'h6715;
                8'd135:origin<=15'h6715;
                8'd136:origin<=15'h5fff;
                8'd137:origin<=15'h5fff;
                8'd138:origin<=15'h5fff;
                8'd139:origin<=15'h5fff;
                default: origin<=15'h5fff;

            endcase
        end
    end

endmodule
```

5.4 seg_sym_sub.v

```
module seg_sym_sub(
        input [2:0] state,
        output reg [6:0] sym
        );
always @(*)

case(state)
3'b000: sym = 7'b1111111;
3'b001: sym = 7'b0111111;
3'b010: sym = 7'b0011111;
3'b011: sym = 7'b0001111;
3'b100: sym = 7'b0000111;
3'b101: sym = 7'b0000011;
3'b110: sym = 7'b0000001;
3'b111: sym = 7'b0000000;
default: sym = 7'b1111111;
endcase

endmodule
```

5.5 seg_dig_sub.v

```
module seg_dig_sub(
        input [3:0] num,
        output reg [6:0] dig
        );

always @(*)
     case(num)
          0: dig = 7'b0000001;
          1: dig = 7'b1001111;
          2: dig = 7'b0010010;
          3: dig = 7'b0000110;
          4: dig = 7'b1001100;
          5: dig = 7'b0100100;
          6: dig = 7'b0100000;
          7: dig = 7'b0001111;
          8: dig = 7'b0000000;
          9: dig = 7'b0000100;
          'hA: dig = 7'b0001000;
          'hB: dig = 7'b1100000;
          'hC: dig = 7'b0110001;
          'hD: dig = 7'b1000010;
          'hE: dig = 7'b0110000;
          'hF: dig = 7'b0111000;
          default: dig = 7'b0000001;
     endcase
```

```
endmodule
```

5.6 LED_display.v

```
module LED_display(
        input [6:0] dig1,
        input [6:0] sym1,
        input [6:0] dig2,
        input [6:0] sym2,
        input [6:0] dig3,
        input [6:0] sym3,
        //input [6:0] dig4,
        input [6:0] sym4,
        input [11:0] num,
        input CLK_50M,
        input [1:0] LED_ctr1, //0 - on_off, 1 - pause
        input [3:0] LED_ctr2, //0 - ledlit1, 1 - ledlit2, 2 - ledlit3, 3 - ledlit4
        input score,//dig_sym
        output [3:0] an,
        output [6:0] LED
        );

reg [3:0] LEDl;
reg [6:0] LEDs;
reg [25:0] s;

assign LED = LEDs;

always @(posedge CLK_50M )
begin
     if(CLK_50M)
       begin
       s = s + 1 ;
       if(s[25])
       s = 0 ;
       end
end

//扫描检测计数
assign an[0] = (s[15] | s[14] | LEDl[0] | LED_ctr2[0]);
assign an[3] = ((~s[15])|(~s[14]) | LEDl[3] | LED_ctr2[3]) ;
assign an[2] = (~s[15] | s[14] | LEDl[2] | LED_ctr2[2]) ;
assign an[1] = (s[15] |~ s[14] | LEDl[1] | LED_ctr2[1]) ;

always @(posedge CLK_50M)
     begin
          if(LED_ctr1[0]) begin
               LEDl = 4'b0000;
```

```
                if(score) begin
                    case (s[15:14])
                        3:LEDs = dig1;
                        2:LEDs = dig2;
                        1: begin
                        if (num[11] == 1)
                            LEDs = 7'b1111110;
                        else
                            LEDs = 7'b1111111;//dig3;
                        end
                        0:LEDs = 7'b0100100;
                    endcase
                end
                else begin
                    if(LED_ctr1[1]) begin
                        LEDs = 7'b1111110;
                        end
                    else begin
                        case (s[15:14])
                            3:LEDs = sym1;
                            2:LEDs = sym2;
                            1:LEDs = sym3;
                            0:LEDs = sym4;
                        endcase
                    end
                end
            end
            else
                LEDl = 4'b1111;
        end

endmodule
```