2024 寒假在家一起练

学　校：　北京理工大学

专　业：　计算机科学与技术

姓　名：　丘绎楦

平　台：　小脚丫 FPGA 套件 STEP BaseBoard V4.0

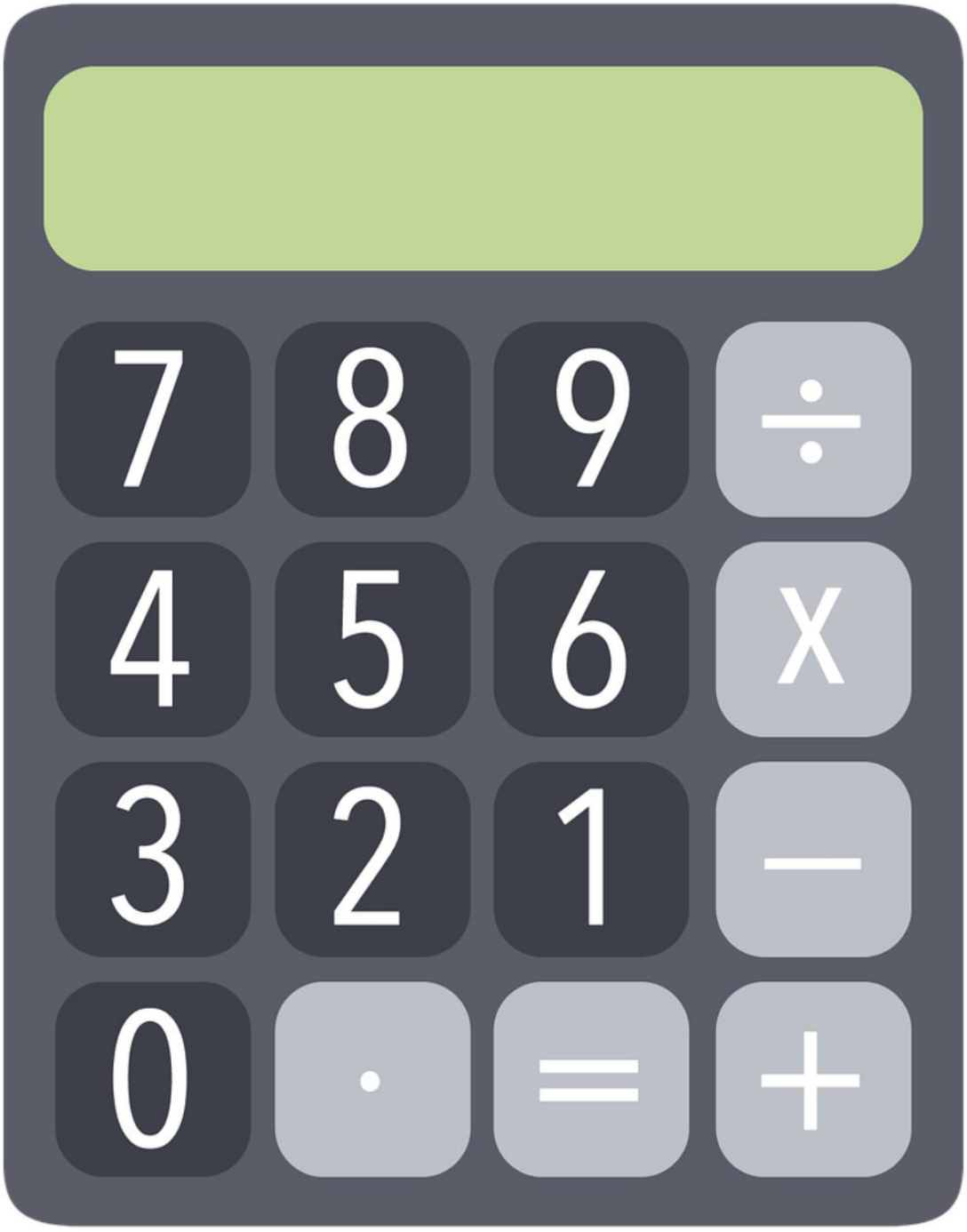
任　务：　两位十进制加、减、乘、除计算器

2024-3-7

目录

1. 项目需求

实现一个两位十进制数加、减、乘、除运算的计算器，运算数和运算符（加、减、乘、除）由按键来控制，4×4 键盘按键分配如下图所示。



运算数和计算结果通过 8 个八段数码管显示。每个运算数使用两个数码管显示，左侧显示十位数，右侧显示个位数。输入两位十进制数时，最高位先在右侧显示，然后其跳变到左侧的数码管上，低位在刚才高位占据的数码管上显示。

2. 需求分析

- **功能需求：**

实现两位十进制数的加法、减法、乘法、除法运算。通过 4×4 键盘进行输入，包括数字键（0-9）和运算符键（加、减、乘、除）。运算数和计算结果通过 8 个八段数码管进行显示。

每个运算数使用两个数码管显示，分别表示十位数和个位数。输入两位十进制数时，最高位先在右侧显示，然后跳变到左侧的数码管上，低位在刚才高位占据的数码管上显示。

- **输入需求：**

用户通过 4×4 键盘输入数字和运算符。支持数字输入（0-9）和运算符输入（加、减、乘、除）。支持连续输入两位十进制数进行运算（包括小数）。输入的数字和运算符需要进行有效性检查，确保输入的格式正确。

- **显示需求：**

使用 8 个八段数码管进行显示，分成两组，每组用于显示一个两位十进制数或计算结果。每个运算数使用两个数码管显示，左侧显示十位数，右侧显示

个位数。运算结构根据是否是小数判断，如果是小数就保留两位小数，否则显示整数。

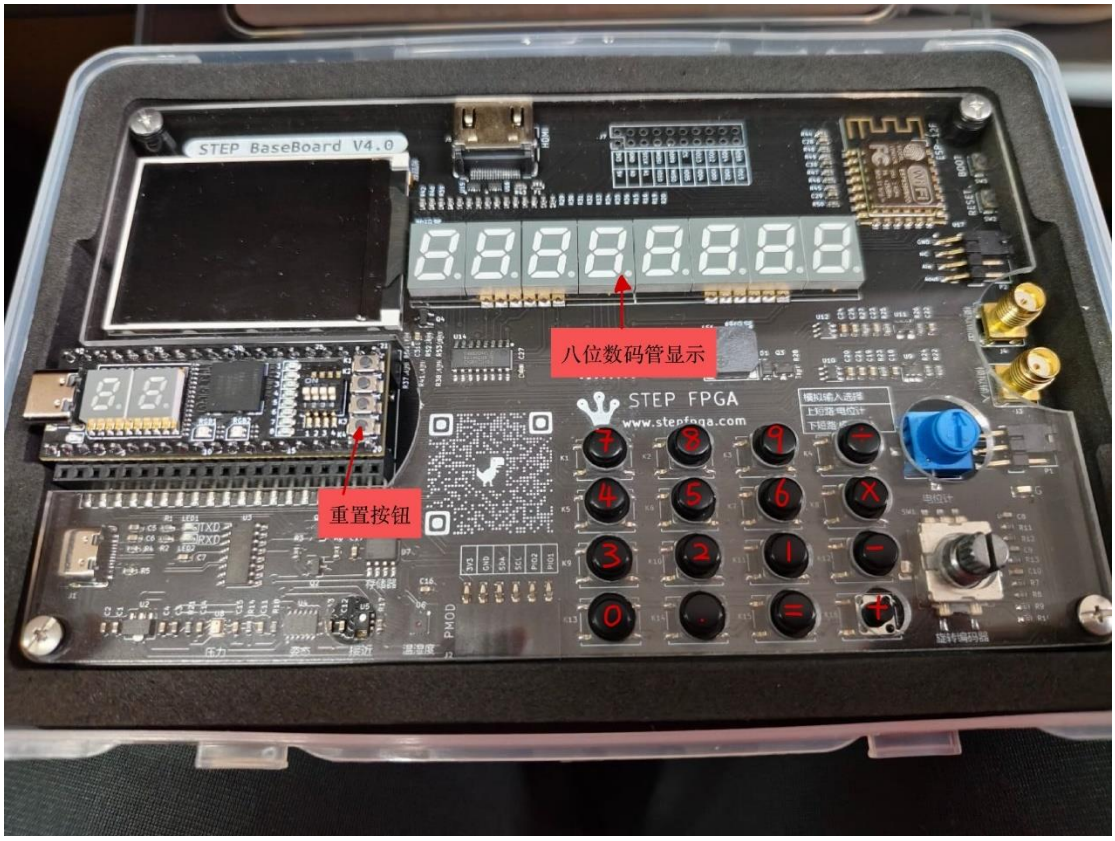
- **运算需求：**

支持加法、减法、乘法、除法四种基本运算。进行运算时，根据用户输入的运算符进行相应的计算。运算过程中需要处理运算数的进位、借位和除法中的除数为零等异常情况。

3. 实现的方式

首先在 array_keyboard 模块中扫描矩阵键盘并检测按键是否被按下。然后将结果放进 key_decode 进行处理，这个模块将按键按下的动作脉冲信号解码成对应的按键值，并标记按键的类型（数字、运算符等）。获得按键的值后便可以对值进行算数运算，cal 模块是主要的计算核心，他接受按键解码模块传来的按键值和案件类型并进行相应的计算逻辑。产生相应的结果（一串二进制数）。为了将此结果显示出来，需要将二进制数转换成 BCD 格式，在 bin2bcd 模块中实现了此功能。最后在 segment_scan 模块中控制数码管的扫描和显示。

4. 功能框图



5. 代码及说明

Calculator.v

该模块负责整合上述所有模块，实现功能的统一执行。

```
module type_system(
    input               clk,            // 系统时钟
    input               rst_n,          // 重置按钮
    input       [3:0]   col,            // 矩阵键盘
    output      [3:0]   row,            // 矩阵键盘
    output              seg_rck,        // 74HC595 的 RCK 管脚
    output              seg_sck,        // 74HC595 的 SCK 管脚
    output              seg_din         // 74HC595 的 SER 管脚
```

```
14. );
15.
16. wire            [15:0]      key_out;
17. wire            [15:0]      key_pulse;  // 脉冲信号
18. wire            [3:0]       seg_data;   // 记录点击的按键对应的值
19. wire            [26:0]     sshow;      // 记录二进制运算的 bin_code
20. wire            [3:0]       statu;      // 标记点击的是数字或运算符等
21. wire            [31:0]      bcd_disp;   // 记录最终显示
22. wire                        un_change;  // 标记是否要进行转换
23. wire            [7:0]       dat_en;     // 数码管数据位显示使能
24. wire            [7:0]       dot_en;     // 数码管小数点位显示使能
25. wire            [3:0]       minus;      // 标记结果是否是负数
26. wire            [7:0]       temp_dat;
27.
28. // 矩阵键盘的扫描和按键检测
29. array_keyboard u1(
30.         .clk            (clk        ),
31.         .rst_n          (rst_n      ),
32.         .col            (col        ),
33.         .row            (row        ),
34.         .key_out        (key_out    ),
35.         .key_pulse      (key_pulse  )
36. );
37.
38.
39. // 将按键的按下动作脉冲信号解码成对应的按键值
40. key_decode u2(
41.         .clk            (clk        ),
42.         .rst_n          (rst_n      ),
43.         .key_pulse      (key_pulse  ),
44.         .seg_data       (seg_data   ),
45.         .statu          (statu      )
46. );
47.
48.
49. // 主要计算模块
50. cal u3(
51.         .clk            (clk        ),
52.         .rst_n          (rst_n      ),
53.         .seg_data       (seg_data   ),
54.         .statu          (statu      ),
55.         .sshow          (sshow      ),
56.         .un_change      (un_change  ),
```

```
57.         .dat_en         (temp_dat   ),
58.         .dot_en         (dot_en     ),
59.         .minus          (minus      )
60. );
61.
62.
63. // 将输入的二进制数转换为 BCD（二进制码十进制）格式
64. bin2bcd u4(
65.         .rst_n          (rst_n      ),
66.         .un_change      (un_change  ),
67.         .bin_code       (sshow      ),
68.         .minus          (minus      ),
69.         .dat_en         (temp_dat   ),
70.         .out_dat_en     (dat_en     ),
71.         .bcd_code       (bcd_disp   )
72. );
73.
74. // 实现数码管的扫描和控制，控制数码管显示的数据和小数点的显示
75. segment_scan u5(
76.         .clk            (clk                ),  //系统时钟 12MHz
77.         .rst_n          (rst_n              ),  //系统复位 低有效
78.         .dat_1          (4'd10              ),  //SEG1 显示的数据输入
79.         .dat_2          (bcd_disp[27:24]    ),  //SEG2 显示的数据输入
80.         .dat_3          (bcd_disp[23:20]    ),  //SEG3 显示的数据输入
81.         .dat_4          (bcd_disp[19:16]    ),  //SEG4 显示的数据输入
82.         .dat_5          (bcd_disp[15:12]    ),  //SEG5 显示的数据输入
83.         .dat_6          (bcd_disp[11:8]     ),  //SEG6 显示的数据输入
84.         .dat_7          (bcd_disp[7:4]      ),  //SEG7 显示的数据输入
85.         .dat_8          (bcd_disp[3:0]      ),  //SEG8 显示的数据输入
86.         .dat_en         (dat_en             ),  //数码管数据位显示使能,
    [MSB~LSB]=[SEG1~SEG8]
87.         .dot_en         (dot_en             ),  //数码管小数点位显示使能,
    [MSB~LSB]=[SEG1~SEG8]
88.         .seg_rck        (seg_rck            ),  //74HC595 的 RCK 管脚
89.         .seg_sck        (seg_sck            ),  //74HC595 的 SCK 管脚
90.         .seg_din        (seg_din            )   //74HC595 的 SER 管脚
91. );
92.
93. endmodule
```

array_keyboard.v

该模块的主要功能是对矩阵键盘进行扫描，并检测键盘按键的按下状态，最终输出按键脉冲信号。

```
1.      // 矩阵键盘的扫描和按键检测
2.       module array_keyboard #
3.      (
4.          parameter   CNT_200HZ = 60000
5.      )
6.      (
7.          input                   clk,
8.          input                   rst_n,
9.          input           [3:0]   col,
10.         output reg      [3:0]   row,
11.         output reg      [15:0]  key_out,
12.         output          [15:0]  key_pulse
13.     );
14.
15.         localparam      STATE0 = 2'b00;
16.         localparam      STATE1 = 2'b01;
17.         localparam      STATE2 = 2'b10;
18.         localparam      STATE3 = 2'b11;
19.
20.         //计数器计数分频实现 5ms 周期信号 clk_200hz,系统时钟 12MHz
21.         reg [15:0]  cnt;
22.         reg         clk_200hz;
23.         always@(posedge clk or negedge rst_n) begin  //复位时计数器 cnt 清零，
               clk_200hz 信号起始电平为低电平
24.             if(!rst_n) begin
25.                 cnt <= 16'd0;
26.                 clk_200hz <= 1'b0;
27.             end else begin
28.                 if(cnt >= ((CNT_200HZ>>1) - 1)) begin  //数字逻辑中右移 1 位相当于
               除 2
29.                     cnt <= 16'd0;
30.                     clk_200hz <= ~clk_200hz;  //clk_200hz 信号取反
31.                 end else begin
32.                     cnt <= cnt + 1'b1;
33.                     clk_200hz <= clk_200hz;
34.                 end
```

```
35.             end
36.         end
37.
38.         reg     [1:0]       c_state;
39.         //状态机根据 clk_200hz 信号在 4 个状态间循环，每个状态对矩阵按键的行接口单行有效
40.         always@(posedge clk_200hz or negedge rst_n) begin
41.             if(!rst_n) begin
42.                 c_state <= STATE0;
43.                 row <= 4'b1110;
44.             end else begin
45.                 case(c_state)
46.                     //状态 c_state 跳转及对应状态下矩阵按键的 row 输出
47.                     STATE0: begin c_state <= STATE1; row <= 4'b1101; end
48.                     STATE1: begin c_state <= STATE2; row <= 4'b1011; end
49.                     STATE2: begin c_state <= STATE3; row <= 4'b0111; end
50.                     STATE3: begin c_state <= STATE0; row <= 4'b1110; end
51.                     default:begin c_state <= STATE0; row <= 4'b1110; end
52.                 endcase
53.             end
54.         end
55.
56.         reg [15:0]  key,key_r;
57.         //因为每个状态中单行有效，通过对列接口的电平状态采样得到对应 4 个按键的状态，依次循环
58.         always@(negedge clk_200hz or negedge rst_n) begin
59.             if(!rst_n) begin
60.                 key_out <= 16'hffff; key_r <= 16'hffff; key <= 16'hffff;
61.             end else begin
62.                 case(c_state)
63.                     //采集当前状态的列数据赋值给对应的寄存器位
64.                     //对键盘采样数据进行判定，连续两次采样低电平判定为按键按下
65.                     STATE0: begin key_out[ 3: 0] <= key_r[ 3: 0]|key[ 3: 0]; key_r[ 3: 0] <= key[ 3: 0]; key[ 3: 0] <= col; end
66.                     STATE1: begin key_out[ 7: 4] <= key_r[ 7: 4]|key[ 7: 4]; key_r[ 7: 4] <= key[ 7: 4]; key[ 7: 4] <= col; end
67.                     STATE2: begin key_out[11: 8] <= key_r[11: 8]|key[11: 8]; key_r[11: 8] <= key[11: 8]; key[11: 8] <= col; end
68.                     STATE3: begin key_out[15:12] <= key_r[15:12]|key[15:12]; key_r[15:12] <= key[15:12]; key[15:12] <= col; end
69.                     default:begin key_out <= 16'hffff; key_r <= 16'hffff; key <= 16'hffff; end
70.                 endcase
```

```
            end
        end

        reg     [15:0]      key_out_r;
        always @ ( posedge clk  or  negedge rst_n )
            if (!rst_n) key_out_r <= 16'hffff;
            else begin
                key_out_r <= key_out;   //将前一刻的值延迟锁存
            end

        assign key_pulse= key_out_r & ( ~key_out);   //通过前后两个时刻的值判断

    endmodule
```

key_decode.v

该模块的主要功能是根据按键的动作脉冲信号，将按键值解码，并提供相应的按键状态。

```
    // 将按键的按下动作脉冲信号（key_pulse）解码成对应的按键值
    module key_decode(
        input                   clk,
        input                   rst_n,
        input           [15:0]  key_pulse,      //按键消抖后动作脉冲信号
        output reg      [3:0]   seg_data,       //表示按键按下的值
        output reg      [3:0]   statu
    );


    always@(posedge clk or negedge rst_n) begin
        if(!rst_n) begin
            statu <= 4'd0;
        end else begin
            case(key_pulse)  //key_pulse 脉宽等于 clk_in 的周期
                16'h0001: begin seg_data <= 4'd7; statu <= 4'd1; end
                16'h0002: begin seg_data <= 4'd8; statu <= 4'd1; end
                16'h0004: begin seg_data <= 4'd9; statu <= 4'd1; end
                16'h0008: begin seg_data <= 4'd14;statu <= 4'd2; end
                16'h0010: begin seg_data <= 4'd4; statu <= 4'd1; end
```

```
                16'h0020: begin seg_data <= 4'd5; statu <= 4'd1; end
                16'h0040: begin seg_data <= 4'd6; statu <= 4'd1; end
                16'h0080: begin seg_data <= 4'd13;statu <= 4'd2; end
                16'h0100: begin seg_data <= 4'd3; statu <= 4'd1; end
                16'h0200: begin seg_data <= 4'd2; statu <= 4'd1; end
                16'h0400: begin seg_data <= 4'd1; statu <= 4'd1; end
                16'h0800: begin seg_data <= 4'd12;statu <= 4'd2; end
                16'h1000: begin seg_data <= 4'd0; statu <= 4'd1; end
                16'h2000: begin seg_data <= 4'd15;statu <= 4'd3; end
                16'h4000: begin seg_data <= 4'd10;statu <= 4'd4; end
                16'h8000: begin seg_data <= 4'd11;statu <= 4'd2; end
                default: begin seg_data <= seg_data; statu <= 4'd0; end  //无按键
            按下时保持
             endcase

        end
    end

    endmodule
```

cal.v

这个模块是主要的计算核心，它接收按键解码模块传来的按键值和按键类型，并进行相应的计算逻辑。根据输入的数字和运算符进行运算，并产生相应的输出结果。

```
    // 主要计算模块
    module cal (
            input               clk,
            input               rst_n,
            input       [3:0]   seg_data,   // 记录点击了什么数字或运算符
            input       [3:0]   statu,      // 标记点击的是数字或运算符等
            output reg  [26:0]  sshow,      // 记录二进制运算的 bin_code
            output reg          un_change,  // 标记是否需要进行转换
            output reg  [7:0]   dat_en,     // 数码管数据位显示使能
            output reg  [7:0]   dot_en,     // 数码管小数点位显示使能
            output reg  [3:0]   minus       // 标记结果是否是负数
    );
```

```
13.
14.     reg [3:0]   t_press;        // 点击数字次数
15.     reg [26:0]  num_1;          // 运算数 1
16.     reg [26:0]  num_2;          // 运算数 2
17.     reg [26:0]  result;         // 运算结果
18.     reg [3:0]   op;             // 标记是否点击了运算符
19.     reg         flag;           // 标记是否清空数码管
20.     reg         float_num1;     // 标记运算数 1 是否是小数
21.     reg         float_num2;     // 标记运算数 2 是否是小数
22.     reg [3:0]   press_num1;     // 运算数 1 小数点位置
23.     reg [3:0]   press_num2;     // 运算数 2 小数点位置
24.     reg [3:0]   temp_press;
25.
26.     always@(posedge clk or negedge rst_n) begin
27.         if(!rst_n) begin        // 重置
28.             num_1 <= 27'b0;
29.             num_2 <= 27'b0;
30.             sshow <= 27'b0;
31.             dot_en <= 8'h0;
32.             op <= 4'd0;
33.             flag <= 0;
34.             dat_en <= 8'h0;
35.             t_press <= 0;
36.             float_num1 <= 0;
37.             float_num2 <= 0;
38.             minus <= 0;
39.             press_num1 <= 0;
40.             press_num2 <= 0;
41.         end else begin
42.             if(statu) begin
43.                 case(statu)
44.                     4'd1: begin // 如果点击的是数字
45.                         t_press = t_press + 4'b1;
46.                         dot_en = dot_en << 1;
47.                         case(t_press)                       // 每点击一次多亮一
        位
48.                             4'd1: dat_en <= 8'b0000_0001;
49.                             4'd2: dat_en <= 8'b0000_0011;
50.                             4'd3: dat_en <= 8'b0000_0111;
51.                             4'd4: dat_en <= 8'b0000_1111;
52.                             4'd5: dat_en <= 8'b0001_1111;
53.                             4'd6: dat_en <= 8'b0011_1111;
54.                             4'd7: dat_en <= 8'b0111_1111;
```

```
55.                         endcase
56.                         un_change = 0;
57.                         if(flag == 1) begin
58.                             sshow = 27'd0;
59.                         end
60.                         sshow <= sshow * 10 + seg_data;
61.                         flag = 0;
62.                     end
63.                     4'd2: begin // 如果点击的是运算符
64.                         if(float_num1)
65.                             press_num1 = t_press - temp_press;
66.                         
67.                         t_press = 0;
68.                         dat_en <= 8'b0000_0001;
69.                         dot_en = 8'b0;
70.                         un_change = 1;
71.                         flag = 1;
72.                         num_1 = sshow;
73.                         if(seg_data == 4'd11) begin             // 标记点击的运算
    符及显示该运算符
74.                             op <= 4'd11;
75.                             sshow = 4'd11;
76.                         end else if(seg_data == 4'd12) begin
77.                             op <= 4'd12;
78.                             sshow = 4'd12;
79.                         end else if(seg_data == 4'd13) begin
80.                             op <= 4'd13;
81.                             sshow = 4'd13;
82.                         end else if(seg_data == 4'd14) begin
83.                             op <= 4'd14;
84.                             sshow = 4'd14;
85.                         end
86.                     end
87.                     4'd3: begin // 如果点击的是小数点
88.                         if(t_press) begin
89.                             dot_en <= 8'b0000_0001;
90.                             if(op)
91.                                 float_num2 = 1;
92.                             else
93.                                 float_num1 = 1;
94.                             temp_press = t_press;
95.                         end
96.                     end
```

```
                    4'd4: begin // 如果点击的是等于
                        num_2 = sshow;
                        if(float_num2)
                            press_num2 = t_press - temp_press;
                        if((float_num1 || float_num2) && op != 4'd13) begin
                            if(press_num1 > press_num2) begin
                                case(press_num1 - press_num2)
            // 对齐小数位
                                    4'b0001: num_2 = num_2 * 10;
                                    4'b0010: num_2 = num_2 * 100;
                                    4'b0011: num_2 = num_2 * 1000;
                                endcase
                            end else begin
                                case(press_num2 - press_num1)
                                    4'b0001: num_1 = num_1 * 10;
                                    4'b0010: num_1 = num_1 * 100;
                                    4'b0011: num_1 = num_1 * 1000;
                                endcase
                            end
                        end

                        if(op == 4'd11) begin                                   //
            加法处理
                            result = num_1 + num_2;
                            sshow = result;
                        end else if(op == 4'd12) begin                          //
            减法处理
                            if(num_1 >= num_2)
                                result = num_1 - num_2;
                            else begin
                                result = num_2 - num_1;
                                minus = 4'd1;
                            end
                            sshow = result;
                        end else if(op == 4'd13) begin                          //
            乘法处理
                            result = num_1 * num_2;
                            sshow = result;
                        end else if(op == 4'd14) begin                          //
            除法处理
                            if(num_2 == 0) begin                                //
            如果除数为 0, 显示 xxxx
                                un_change = 1;
```

```
                                sshow = 16'b1101_1101_1101_1101;
                            end else begin
                                result = num_1 * 100 / num_2;
                                dot_en <= 8'b0000_0100;
                                sshow = result;
                            end
                        end

                        if((float_num1 || float_num2) && (op == 4'd11 || op == 4'd12)) begin    // 对其小数位
                            if(press_num1 > press_num2) begin
                                case(press_num1)
                                    4'b0001: dot_en <= 8'b0000_0010;
                                    4'b0010: dot_en <= 8'b0000_0100;
                                    4'b0011: dot_en <= 8'b0000_1000;
                                    4'b0100: dot_en <= 8'b0001_0000;
                                endcase
                            end else begin
                                case(press_num2)
                                    4'b0001: dot_en <= 8'b0000_0010;
                                    4'b0010: dot_en <= 8'b0000_0100;
                                    4'b0011: dot_en <= 8'b0000_1000;
                                    4'b0100: dot_en <= 8'b0001_0000;
                                endcase
                            end
                        end else if((float_num1 || float_num2) && op == 4'd13) begin
                            case(press_num1 + press_num2)
                                4'b0001: dot_en <= 8'b0000_0010;
                                4'b0010: dot_en <= 8'b0000_0100;
                                4'b0011: dot_en <= 8'b0000_1000;
                                4'b0100: dot_en <= 8'b0001_0000;
                            endcase
                        end else if((float_num1 || float_num2) && op == 4'd14) begin
                            dot_en <= 8'b0000_0100;
                            dat_en <= 8'b0000_0111;
                        end

                        if(sshow >= 27'd1000000)                    // 结果显示控制
                            dat_en <= 8'hff;
```

```
                        else if(sshow >= 27'd100000)
                            dat_en <= 8'b1011_1111;
                        else if(sshow >= 27'd10000)
                            dat_en <= 8'b1001_1111;
                        else if(sshow >= 27'd1000)
                            dat_en <= 8'b1000_1111;
                        else if(sshow >= 27'd100)
                            dat_en <= 8'b1000_0111;
                        else if(sshow >= 27'd10 && op != 4'd14)
                            dat_en <= 8'b1000_0011;
                        else if(sshow >= 27'd0 && op != 4'd14)
                            dat_en <= 8'b1000_0001;
                    end
                endcase
            end
        end
    end

    endmodule
```

bin2bcd.v

该模块的主要功能是实现二进制数到 BCD 码的转换，并根据负数标记和数码管数据位显示使能信号，控制输出的 BCD 码和数码管显示使能信号。

```
    // 将输入的二进制数转换为 BCD（二进制码十进制）格式
    module bin2bcd(
        input               rst_n,
        input               un_change,      // 标记是否要进行转换
        input       [26:0]  bin_code,       // 需要转换成 bcd 的二进制
        input       [3:0]   minus,          // 标记结果是否是负数
        input       [7:0]   dat_en,         // 数码管数据位显示使能
        output reg  [7:0]   out_dat_en,     // 返回八位数码管显示
        output reg  [31:0]  bcd_code        // 记录转换后的 bcd_code
    );

    reg [58:0] shift_reg;

    always @ (bin_code or rst_n) begin
```

```
15.         shift_reg = {32'h0, bin_code};
16.         out_dat_en <= dat_en;
17.         if (!rst_n)
18.             bcd_code <= 0;
19.         else begin
20.             if (un_change)
21.                 bcd_code <= bin_code;
22.             else begin
23.                 repeat (27) begin
24.                     if (shift_reg[30:27] >= 5) shift_reg[30:27] = shift_reg[30:2
                7] + 2'b11;
25.                     if (shift_reg[34:31] >= 5) shift_reg[34:31] = shift_reg[34:3
                1] + 2'b11;
26.                     if (shift_reg[38:35] >= 5) shift_reg[38:35] = shift_reg[38:3
                5] + 2'b11;
27.                     if (shift_reg[42:39] >= 5) shift_reg[42:39] = shift_reg[42:3
                9] + 2'b11;
28.                     if (shift_reg[46:43] >= 5) shift_reg[46:43] = shift_reg[46:4
                3] + 2'b11;
29.                     if (shift_reg[50:47] >= 5) shift_reg[50:47] = shift_reg[50:4
                7] + 2'b11;
30.                     if (shift_reg[54:51] >= 5) shift_reg[54:51] = shift_reg[54:5
                1] + 2'b11;
31.                     if (shift_reg[58:55] >= 5) shift_reg[58:55] = shift_reg[58:5
                5] + 2'b11;
32.                     shift_reg = shift_reg << 1;
33.                 end
34.                 if(minus) begin     // 如果结果是负数，在对应位置加上负号
35.                     case(dat_en)
36.                         8'b1000_0001: begin shift_reg[34:31] = 4'd12; out_dat_en
                 <= 8'b1000_0011; end
37.                         8'b1000_0011: begin shift_reg[38:35] = 4'd12; out_dat_en
                 <= 8'b1000_0111; end
38.                         8'b1000_0111: begin shift_reg[42:39] = 4'd12; out_dat_en
                 <= 8'b1000_1111; end
39.                         8'b1000_1111: begin shift_reg[46:43] = 4'd12; out_dat_en
                 <= 8'b1001_1111; end
40.                         8'b1001_1111: begin shift_reg[50:47] = 4'd12; out_dat_en
                 <= 8'b1011_1111; end
41.                         8'b1011_1111: begin shift_reg[54:51] = 4'd12; out_dat_en
                 <= 8'b1111_1111; end
42.                     endcase
43.                 end
```

```
                bcd_code <= shift_reg[58:27];
            end
        end
    end

    endmodule
```

segment_scan.v

该模块的主要功能是对数码管的数据进行扫描和控制，根据外部输入的数据和使能信号，完成数码管的显示操作。

```
    // 实现数码管的扫描和控制，控制数码管显示的数据和小数点的显示
    module segment_scan(
            input               clk,        //系统时钟 12MHz
            input               rst_n,      //系统复位 低有效
            input       [3:0]   dat_1,      //SEG1 显示的数据输入
            input       [3:0]   dat_2,      //SEG2 显示的数据输入
            input       [3:0]   dat_3,      //SEG3 显示的数据输入
            input       [3:0]   dat_4,      //SEG4 显示的数据输入
            input       [3:0]   dat_5,      //SEG5 显示的数据输入
            input       [3:0]   dat_6,      //SEG6 显示的数据输入
            input       [3:0]   dat_7,      //SEG7 显示的数据输入
            input       [3:0]   dat_8,      //SEG8 显示的数据输入
            input       [7:0]   dat_en,     //数码管数据位显示使能
            input       [7:0]   dot_en,     //数码管小数点位显示使能
            output  reg         seg_rck,    //74HC595 的 RCK 管脚
            output  reg         seg_sck,    //74HC595 的 SCK 管脚
            output  reg         seg_din     //74HC595 的 SER 管脚
    );

    localparam  CNT_40KHz = 300;    //分频系数

    localparam  IDLE    =   3'd0;
    localparam  MAIN    =   3'd1;
    localparam  WRITE   =   3'd2;
    localparam  LOW     =   1'b0;
    localparam  HIGH    =   1'b1;

```

```
28.     //创建数码管的字库，字库数据依段码顺序有关
29.     reg[6:0] seg [15:0];
30.     always @(negedge rst_n) begin
31.         seg[0]  =   7'h3f;   // 0
32.         seg[1]  =   7'h06;   // 1
33.         seg[2]  =   7'h5b;   // 2
34.         seg[3]  =   7'h4f;   // 3
35.         seg[4]  =   7'h66;   // 4
36.         seg[5]  =   7'h6d;   // 5
37.         seg[6]  =   7'h7d;   // 6
38.         seg[7]  =   7'h07;   // 7
39.         seg[8]  =   7'h7f;   // 8
40.         seg[9]  =   7'h6f;   // 9
41.         seg[10] =   7'b100_1000;   // =
42.         seg[11] =   7'b100_0110;   // +
43.         seg[12] =   7'b100_0000;   // -
44.         seg[13] =   7'b111_0110;   // *
45.         seg[14] =   7'b100_1001;   // /
46.     end
47.
48.     //计数器对系统时钟信号进行计数
49.     reg [9:0] cnt = 1'b0;
50.     always@(posedge clk or negedge rst_n) begin
51.         if(!rst_n) cnt <= 1'b0;
52.         else if(cnt>=(CNT_40KHz-1)) cnt <= 1'b0;
53.         else cnt <= cnt + 1'b1;
54.     end
55.
56.     //根据计数器计数的周期产生分频的脉冲信号
57.     reg clk_40khz = 1'b0;
58.     always@(posedge clk or negedge rst_n) begin
59.         if(!rst_n) clk_40khz <= 1'b0;
60.         else if(cnt<(CNT_40KHz>>1)) clk_40khz <= 1'b0;
61.         else clk_40khz <= 1'b1;
62.     end
63.
64.     //使用状态机完成数码管的扫描和 74HC595 时序的实现
65.     reg     [15:0]      data;
66.     reg     [2:0]       cnt_main;
67.     reg     [5:0]       cnt_write;
68.     reg     [2:0]       state = IDLE;
69.     always@(posedge clk_40khz or negedge rst_n) begin
70.
```

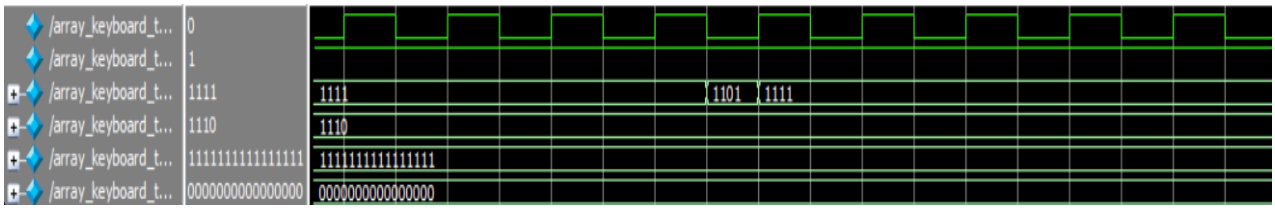
```
71.         if(!rst_n) begin    //复位状态下，各寄存器置初值
72.             state <= IDLE;
73.             cnt_main <= 3'd0; cnt_write <= 6'd0;
74.             seg_din <= 1'b0; seg_sck <= LOW; seg_rck <= LOW;
75.         end else begin
76.             case(state)
77.                 IDLE:begin  //IDLE 作为第一个状态，相当于软复位
78.                         state <= MAIN;
79.                         cnt_main <= 3'd0; cnt_write <= 6'd0;
80.                         seg_din <= 1'b0; seg_sck <= LOW; seg_rck <= LOW;
81.                     end
82.                 MAIN:begin
83.                         cnt_main <= cnt_main + 1'b1;
84.                         state <= WRITE;       //在配置完发给 74HC595 的数据同时跳转至
               WRITE 状态，完成串行时序
85.
86.
87.                         case(cnt_main)
88.                             //对 8 位数码管逐位扫描
89.                             //data           [15:8]为段选，           [7:0]为位选
90.                             3'd0: data <= {{dot_en[7],seg[dat_1]},dat_en[7]?8'hf
               e:8'hff};
91.                             3'd1: data <= {{dot_en[6],seg[dat_2]},dat_en[6]?8'hf
               d:8'hff};
92.                             3'd2: data <= {{dot_en[5],seg[dat_3]},dat_en[5]?8'hf
               b:8'hff};
93.                             3'd3: data <= {{dot_en[4],seg[dat_4]},dat_en[4]?8'hf
               7:8'hff};
94.                             3'd4: data <= {{dot_en[3],seg[dat_5]},dat_en[3]?8'he
               f:8'hff};
95.                             3'd5: data <= {{dot_en[2],seg[dat_6]},dat_en[2]?8'hd
               f:8'hff};
96.                             3'd6: data <= {{dot_en[1],seg[dat_7]},dat_en[1]?8'hb
               f:8'hff};
97.                             3'd7: data <= {{dot_en[0],seg[dat_8]},dat_en[0]?8'h7
               f:8'hff};
98.                             default: data <= {8'h00,8'hff};
99.                         endcase
100.                    end
101.                WRITE:begin
102.                        if(cnt_write >= 6'd33) cnt_write <= 1'b0;
103.                        else cnt_write <= cnt_write + 1'b1;
104.                        case(cnt_write)
```

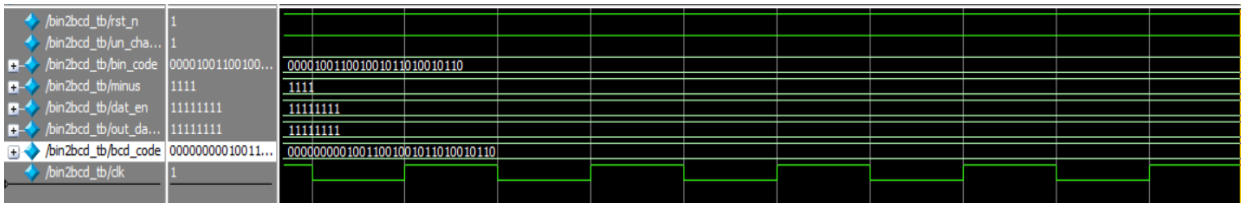
```
105.                        //74HC595 是串行转并行的芯片，3 路输入可产生 8 路输出，而且可以级联使用
106.                        //74HC595 的时序实现，参考 74HC595 的芯片手册
107.                        6'd0:  begin seg_sck <= LOW; seg_din <= data[15]; end      //SCK 下降沿时 SER 更新数据
108.                        6'd1:  begin seg_sck <= HIGH; end                          //SCK 上升沿时 SER 数据稳定
109.                        6'd2:  begin seg_sck <= LOW; seg_din <= data[14]; end
110.                        6'd3:  begin seg_sck <= HIGH; end
111.                        6'd4:  begin seg_sck <= LOW; seg_din <= data[13]; end
112.                        6'd5:  begin seg_sck <= HIGH; end
113.                        6'd6:  begin seg_sck <= LOW; seg_din <= data[12]; end
114.                        6'd7:  begin seg_sck <= HIGH; end
115.                        6'd8:  begin seg_sck <= LOW; seg_din <= data[11]; end
116.                        6'd9:  begin seg_sck <= HIGH; end
117.                        6'd10: begin seg_sck <= LOW; seg_din <= data[10]; end
118.                        6'd11: begin seg_sck <= HIGH; end
119.                        6'd12: begin seg_sck <= LOW; seg_din <= data[9]; end
120.                        6'd13: begin seg_sck <= HIGH; end
121.                        6'd14: begin seg_sck <= LOW; seg_din <= data[8]; end
122.                        6'd15: begin seg_sck <= HIGH; end
123.                        6'd16: begin seg_sck <= LOW; seg_din <= data[7]; end
124.                        6'd17: begin seg_sck <= HIGH; end
125.                        6'd18: begin seg_sck <= LOW; seg_din <= data[6]; end
126.                        6'd19: begin seg_sck <= HIGH; end
127.                        6'd20: begin seg_sck <= LOW; seg_din <= data[5]; end
128.                        6'd21: begin seg_sck <= HIGH; end
129.                        6'd22: begin seg_sck <= LOW; seg_din <= data[4]; end
130.                        6'd23: begin seg_sck <= HIGH; end
131.                        6'd24: begin seg_sck <= LOW; seg_din <= data[3]; end
132.                        6'd25: begin seg_sck <= HIGH; end
```

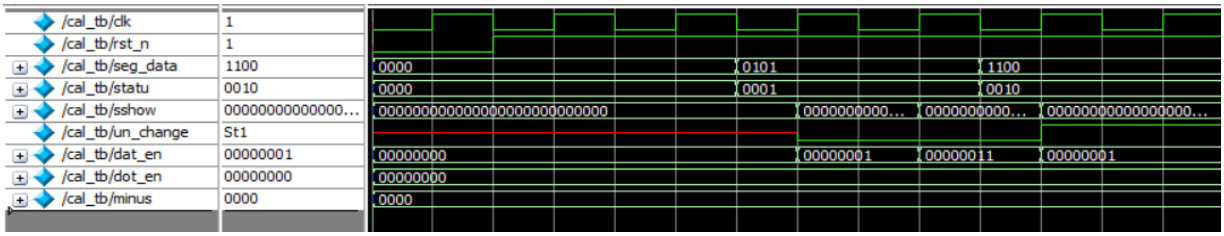
```
                            6'd26: begin seg_sck <= LOW; seg_din <= data[2]; end
                            6'd27: begin seg_sck <= HIGH; end
                            6'd28: begin seg_sck <= LOW; seg_din <= data[1]; end
                            6'd29: begin seg_sck <= HIGH; end
                            6'd30: begin seg_sck <= LOW; seg_din <= data[0]; end
                            6'd31: begin seg_sck <= HIGH; end
                            6'd32: begin seg_rck <= HIGH; end
                        //当 16 位数据传送完成后 RCK 拉高，输出生效
                            6'd33: begin seg_rck <= LOW; state <= MAIN; end
                            default: ;
                        endcase
                    end
                default: state <= IDLE;
            endcase
        end
    end

    endmodule
```

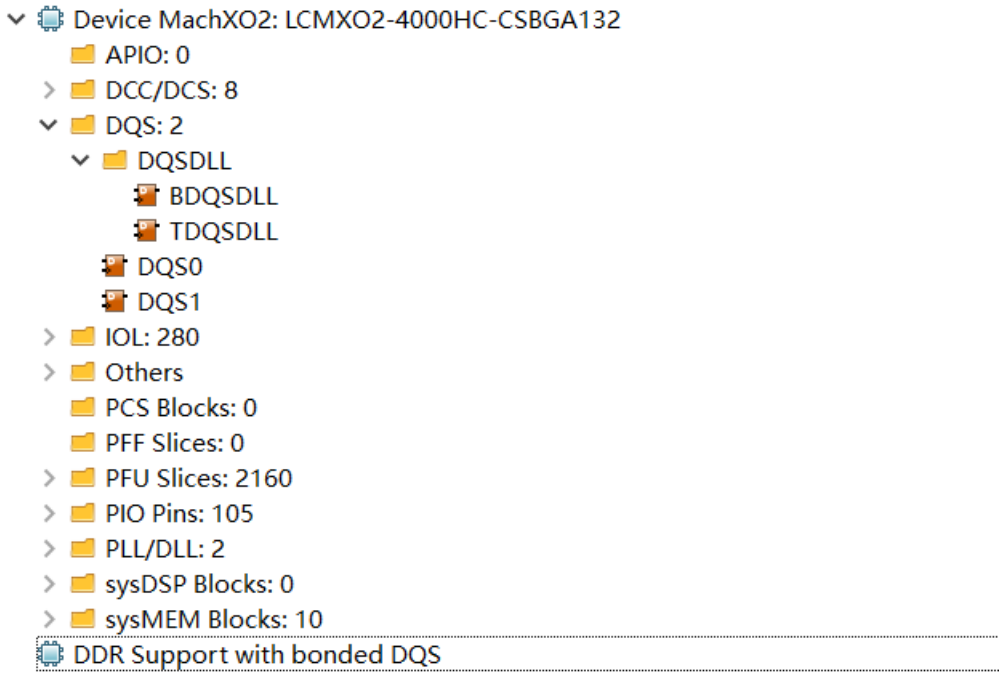
仿真波形图







FPGA 的资源利用说明



演示视频

https://www.bilibili.com/video/BV1py421B7JH/

<iframe src="//player.bilibili.com/player.html?aid=155129 4792&bvid=BV1py421B7JH&cid=1461926353&p =1" scrolling="no" border="0" frameborder="no" framespacing="0" allowfullscreen="true"> </iframe>