**2024寒假在家一起练**

北京理工大学

计算机科学与技术

丘绎楦

小脚丫FPGA套件STEP BaseBoard V4.0

两位十进制加、减、乘、除计算器

学 校：

专 业：

姓 名：

平 台：

任 务：

2024-3-7

**目录**

1. 项目需求………………………………………………………………..2
2. 需求分析………………………………………………………………..3
3. 实现的方式…………………………………………………………….4
4. 功能框图………………………………………………………………..5
5. 代码及说明

Calculator.v…………………………………………………………..5

array\_keyboard.v…………………………………………………..8

key\_decode.v………………………………………………………10

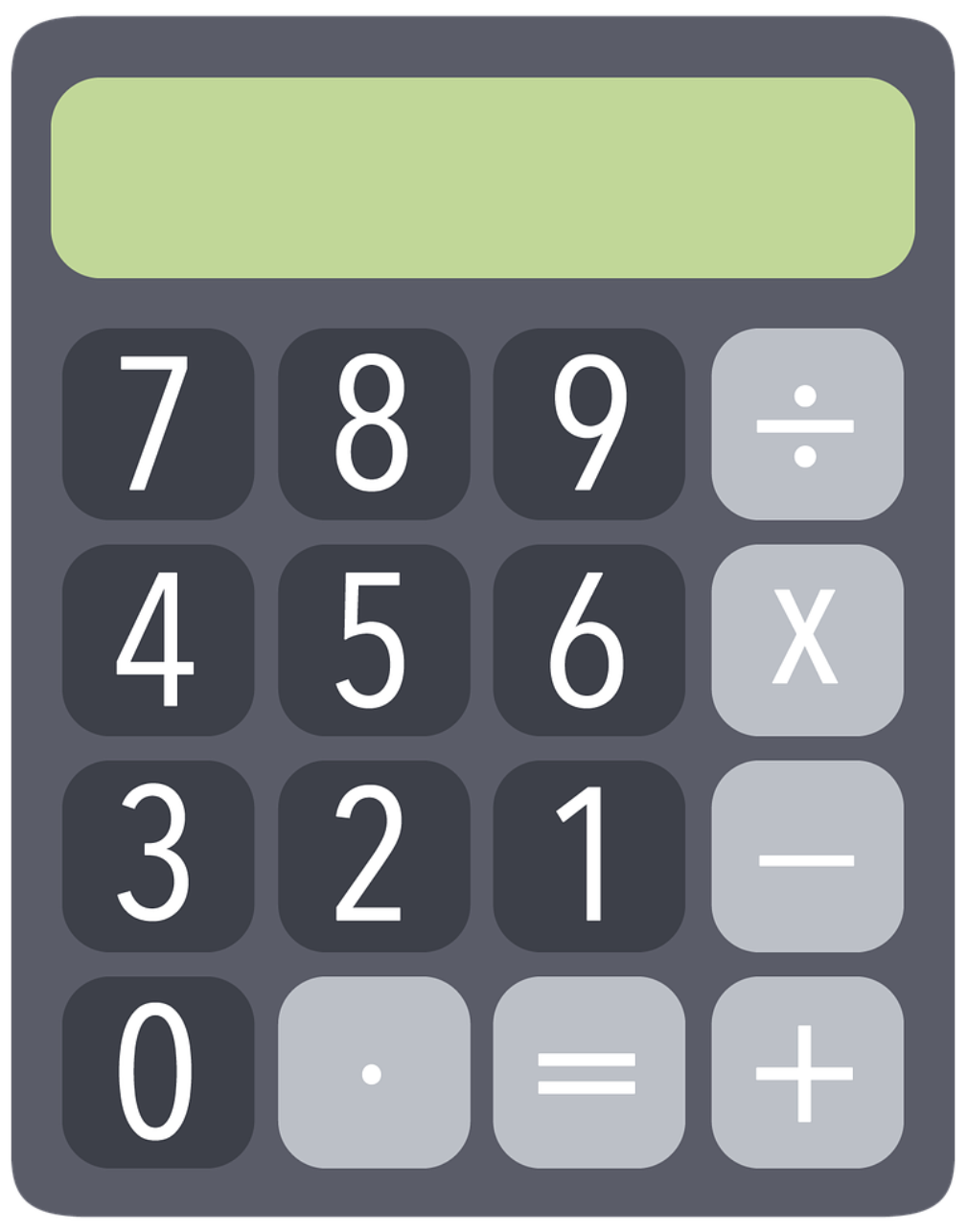
cal.v…………………………………………………………………….11

bin2bcd.v…………………………………………………………….16

segment\_scan.v…………………………………………………..18

1. 仿真波形图…………………………………………………………...22
2. FPGA的资源利用说明…………………………………………...23
3. 演示视频………………………………………………………………23
4. **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。

1. **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。

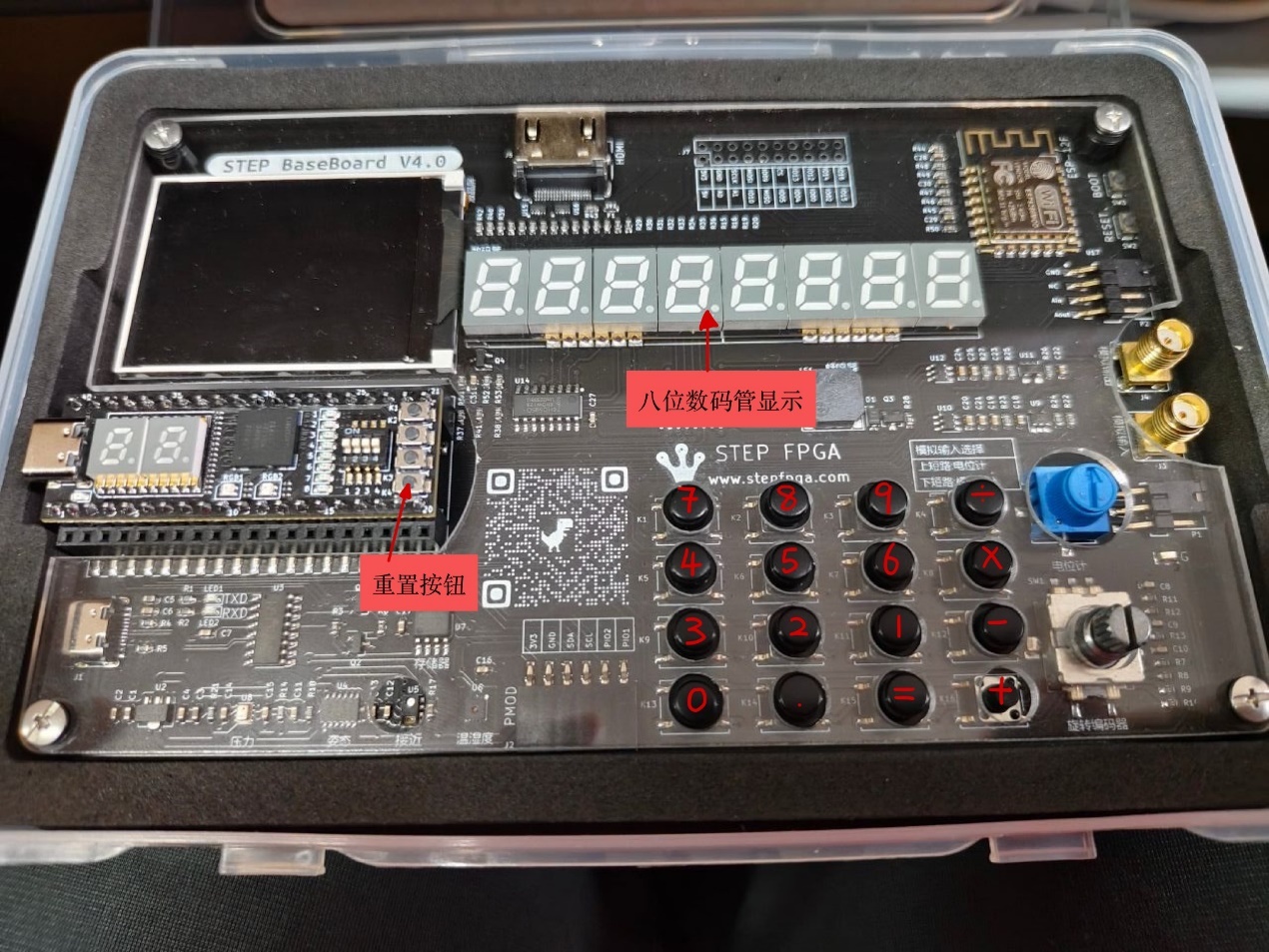
* **运算需求：**

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。

1. **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。

1. **功能框图**



1. **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。

1. module type\_system(
2. input               clk,            // 系统时钟
3. input               rst\_n,          // 重置按钮
4. input       [3:0]   col,            // 矩阵键盘
5. output      [3:0]   row,            // 矩阵键盘
6. output              seg\_rck,        // 74HC595的RCK管脚
7. output              seg\_sck,        // 74HC595的SCK管脚
8. output              seg\_din         // 74HC595的SER管脚
9. );
11. wire            [15:0]      key\_out;
12. wire            [15:0]      key\_pulse;  // 脉冲信号
13. wire            [3:0]       seg\_data;   // 记录点击的按键对应的值
14. wire            [26:0]     sshow;      // 记录二进制运算的bin\_code
15. wire            [3:0]       statu;      // 标记点击的是数字或运算符等
16. wire            [31:0]      bcd\_disp;   // 记录最终显示
17. wire                        un\_change;  // 标记是否要进行转换
18. wire            [7:0]       dat\_en;     // 数码管数据位显示使能
19. wire            [7:0]       dot\_en;     // 数码管小数点位显示使能
20. wire            [3:0]       minus;      // 标记结果是否是负数
21. wire            [7:0]       temp\_dat;
23. // 矩阵键盘的扫描和按键检测
24. array\_keyboard u1(
25. .clk            (clk        ),
26. .rst\_n          (rst\_n      ),
27. .col            (col        ),
28. .row            (row        ),
29. .key\_out        (key\_out    ),
30. .key\_pulse      (key\_pulse  )
31. );

34. // 将按键的按下动作脉冲信号解码成对应的按键值
35. key\_decode u2(
36. .clk            (clk        ),
37. .rst\_n          (rst\_n      ),
38. .key\_pulse      (key\_pulse  ),
39. .seg\_data       (seg\_data   ),
40. .statu          (statu      )
41. );
43. // 主要计算模块
44. cal u3(
45. .clk            (clk        ),
46. .rst\_n          (rst\_n      ),
47. .seg\_data       (seg\_data   ),
48. .statu          (statu      ),
49. .sshow          (sshow      ),
50. .un\_change      (un\_change  ),
51. .dat\_en         (temp\_dat   ),
52. .dot\_en         (dot\_en     ),
53. .minus          (minus      )
54. );

57. // 将输入的二进制数转换为BCD（二进制码十进制）格式
58. bin2bcd u4(
59. .rst\_n          (rst\_n      ),
60. .un\_change      (un\_change  ),
61. .bin\_code       (sshow      ),
62. .minus          (minus      ),
63. .dat\_en         (temp\_dat   ),
64. .out\_dat\_en     (dat\_en     ),
65. .bcd\_code       (bcd\_disp   )
66. );
68. // 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
69. segment\_scan u5(
70. .clk            (clk                ),  //系统时钟 12MHz
71. .rst\_n          (rst\_n              ),  //系统复位 低有效
72. .dat\_1          (4'd10              ),  //SEG1 显示的数据输入
73. .dat\_2          (bcd\_disp[27:24]    ),  //SEG2 显示的数据输入
74. .dat\_3          (bcd\_disp[23:20]    ),  //SEG3 显示的数据输入
75. .dat\_4          (bcd\_disp[19:16]    ),  //SEG4 显示的数据输入
76. .dat\_5          (bcd\_disp[15:12]    ),  //SEG5 显示的数据输入
77. .dat\_6          (bcd\_disp[11:8]     ),  //SEG6 显示的数据输入
78. .dat\_7          (bcd\_disp[7:4]      ),  //SEG7 显示的数据输入
79. .dat\_8          (bcd\_disp[3:0]      ),  //SEG8 显示的数据输入
80. .dat\_en         (dat\_en             ),  //数码管数据位显示使能，[MSB~LSB]=[SEG1~SEG8]
81. .dot\_en         (dot\_en             ),  //数码管小数点位显示使能，[MSB~LSB]=[SEG1~SEG8]
82. .seg\_rck        (seg\_rck            ),  //74HC595的RCK管脚
83. .seg\_sck        (seg\_sck            ),  //74HC595的SCK管脚
84. .seg\_din        (seg\_din            )   //74HC595的SER管脚
85. );
87. 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. );
15. localparam      STATE0 = 2'b00;
16. localparam      STATE1 = 2'b01;
17. localparam      STATE2 = 2'b10;
18. localparam      STATE3 = 2'b11;
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**
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**
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
71. **end**
72. **end**
74. reg     [15:0]      key\_out\_r;
75. always @ ( posedge clk  **or**  negedge rst\_n )
76. **if** (!rst\_n) key\_out\_r <= 16'hffff;
77. **else** **begin**
78. key\_out\_r <= key\_out;   //将前一刻的值延迟锁存
79. **end**
81. assign key\_pulse= key\_out\_r & ( ~key\_out);   //通过前后两个时刻的值判断
83. 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。

1. // 将按键的按下动作脉冲信号（key\_pulse）解码成对应的按键值
2. module key\_decode(
3. input                   clk,
4. input                   rst\_n,
5. input           [15:0]  key\_pulse,      //按键消抖后动作脉冲信号
6. output reg      [3:0]   seg\_data,       //表示按键按下的值
7. output reg      [3:0]   statu
8. );

11. always@(posedge clk **or** negedge rst\_n) **begin**
12. **if**(!rst\_n) **begin**
13. statu <= 4'd0;
14. **end** **else** **begin**
15. **case**(key\_pulse)  //key\_pulse脉宽等于clk\_in的周期
16. 16'h0001: begin seg\_data <= 4'd7; statu <= 4'd1; **end**
17. 16'h0002: begin seg\_data <= 4'd8; statu <= 4'd1; **end**
18. 16'h0004: begin seg\_data <= 4'd9; statu <= 4'd1; **end**
19. 16'h0008: begin seg\_data <= 4'd14;statu <= 4'd2; **end**
20. 16'h0010: begin seg\_data <= 4'd4; statu <= 4'd1; **end**
21. 16'h0020: begin seg\_data <= 4'd5; statu <= 4'd1; **end**
22. 16'h0040: begin seg\_data <= 4'd6; statu <= 4'd1; **end**
23. 16'h0080: begin seg\_data <= 4'd13;statu <= 4'd2; **end**
24. 16'h0100: begin seg\_data <= 4'd3; statu <= 4'd1; **end**
25. 16'h0200: begin seg\_data <= 4'd2; statu <= 4'd1; **end**
26. 16'h0400: begin seg\_data <= 4'd1; statu <= 4'd1; **end**
27. 16'h0800: begin seg\_data <= 4'd12;statu <= 4'd2; **end**
28. 16'h1000: begin seg\_data <= 4'd0; statu <= 4'd1; **end**
29. 16'h2000: begin seg\_data <= 4'd15;statu <= 4'd3; **end**
30. 16'h4000: begin seg\_data <= 4'd10;statu <= 4'd4; **end**
31. 16'h8000: begin seg\_data <= 4'd11;statu <= 4'd2; **end**
32. default: **begin** seg\_data <= seg\_data; statu <= 4'd0; **end**  //无按键按下时保持
33. endcase
35. **end**
36. **end**
38. 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。

1. // 主要计算模块
2. module cal (
3. input               clk,
4. input               rst\_n,
5. input       [3:0]   seg\_data,   // 记录点击了什么数字或运算符
6. input       [3:0]   statu,      // 标记点击的是数字或运算符等
7. output reg  [26:0]  sshow,      // 记录二进制运算的bin\_code
8. output reg          un\_change,  // 标记是否需要进行转换
9. output reg  [7:0]   dat\_en,     // 数码管数据位显示使能
10. output reg  [7:0]   dot\_en,     // 数码管小数点位显示使能
11. output reg  [3:0]   minus       // 标记结果是否是负数
12. );
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;
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;
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**
97. 4'd4: **begin** // 如果点击的是等于
98. num\_2 = sshow;
99. **if**(float\_num2)
100. press\_num2 = t\_press - temp\_press;
101. **if**((float\_num1 || float\_num2) && op != 4'd13) **begin**
102. **if**(press\_num1 > press\_num2) **begin**
103. **case**(press\_num1 - press\_num2)                   // 对齐小数位
104. 4'b0001: num\_2 = num\_2 \* 10;
105. 4'b0010: num\_2 = num\_2 \* 100;
106. 4'b0011: num\_2 = num\_2 \* 1000;
107. endcase
108. **end** **else** **begin**
109. **case**(press\_num2 - press\_num1)
110. 4'b0001: num\_1 = num\_1 \* 10;
111. 4'b0010: num\_1 = num\_1 \* 100;
112. 4'b0011: num\_1 = num\_1 \* 1000;
113. endcase
114. **end**
115. **end**
117. **if**(op == 4'd11) **begin**                               // 加法处理
118. result = num\_1 + num\_2;
119. sshow = result;
120. **end** **else** **if**(op == 4'd12) **begin**                      // 减法处理
121. **if**(num\_1 >= num\_2)
122. result = num\_1 - num\_2;
123. **else** **begin**
124. result = num\_2 - num\_1;
125. minus = 4'd1;
126. **end**
127. sshow = result;
128. **end** **else** **if**(op == 4'd13) **begin**                      // 乘法处理
129. result = num\_1 \* num\_2;
130. sshow = result;
131. **end** **else** **if**(op == 4'd14) **begin**                      // 除法处理
132. **if**(num\_2 == 0) **begin**                            // 如果除数为0, 显示xxxx
133. un\_change = 1;
134. sshow = 16'b1101\_1101\_1101\_1101;
135. **end** **else** **begin**
136. result = num\_1 \* 100 / num\_2;
137. dot\_en <= 8'b0000\_0100;
138. sshow = result;
139. **end**
140. **end**
142. **if**((float\_num1 || float\_num2) && (op == 4'd11 || op == 4'd12)) **begin**    // 对其小数位
143. **if**(press\_num1 > press\_num2) **begin**
144. **case**(press\_num1)
145. 4'b0001: dot\_en <= 8'b0000\_0010;
146. 4'b0010: dot\_en <= 8'b0000\_0100;
147. 4'b0011: dot\_en <= 8'b0000\_1000;
148. 4'b0100: dot\_en <= 8'b0001\_0000;
149. endcase
150. **end** **else** **begin**
151. **case**(press\_num2)
152. 4'b0001: dot\_en <= 8'b0000\_0010;
153. 4'b0010: dot\_en <= 8'b0000\_0100;
154. 4'b0011: dot\_en <= 8'b0000\_1000;
155. 4'b0100: dot\_en <= 8'b0001\_0000;
156. endcase
157. **end**
158. **end** **else** **if**((float\_num1 || float\_num2) && op == 4'd13) **begin**
159. **case**(press\_num1 + press\_num2)
160. 4'b0001: dot\_en <= 8'b0000\_0010;
161. 4'b0010: dot\_en <= 8'b0000\_0100;
162. 4'b0011: dot\_en <= 8'b0000\_1000;
163. 4'b0100: dot\_en <= 8'b0001\_0000;
164. endcase
165. **end** **else** **if**((float\_num1 || float\_num2) && op == 4'd14) **begin**
166. dot\_en <= 8'b0000\_0100;
167. dat\_en <= 8'b0000\_0111;
168. **end**

171. **if**(sshow >= 27'd1000000)                    // 结果显示控制
172. dat\_en <= 8'hff;
173. **else** **if**(sshow >= 27'd100000)
174. dat\_en <= 8'b1011\_1111;
175. **else** **if**(sshow >= 27'd10000)
176. dat\_en <= 8'b1001\_1111;
177. **else** **if**(sshow >= 27'd1000)
178. dat\_en <= 8'b1000\_1111;
179. **else** **if**(sshow >= 27'd100)
180. dat\_en <= 8'b1000\_0111;
181. **else** **if**(sshow >= 27'd10 && op != 4'd14)
182. dat\_en <= 8'b1000\_0011;
183. **else** **if**(sshow >= 27'd0 && op != 4'd14)
184. dat\_en <= 8'b1000\_0001;
185. **end**
186. endcase
187. **end**
188. **end**
189. **end**
191. 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。

1. // 将输入的二进制数转换为BCD（二进制码十进制）格式
2. module bin2bcd(
3. input               rst\_n,
4. input               un\_change,      // 标记是否要进行转换
5. input       [26:0]  bin\_code,       // 需要转换成bcd的二进制
6. input       [3:0]   minus,          // 标记结果是否是负数
7. input       [7:0]   dat\_en,         // 数码管数据位显示使能
8. output reg  [7:0]   out\_dat\_en,     // 返回八位数码管显示
9. output reg  [31:0]  bcd\_code        // 记录转换后的bcd\_code
10. );
12. reg [58:0] shift\_reg;
14. 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[30:27] + 2'b11;
25. **if** (shift\_reg[34:31] >= 5) shift\_reg[34:31] = shift\_reg[34:31] + 2'b11;
26. **if** (shift\_reg[38:35] >= 5) shift\_reg[38:35] = shift\_reg[38:35] + 2'b11;
27. **if** (shift\_reg[42:39] >= 5) shift\_reg[42:39] = shift\_reg[42:39] + 2'b11;
28. **if** (shift\_reg[46:43] >= 5) shift\_reg[46:43] = shift\_reg[46:43] + 2'b11;
29. **if** (shift\_reg[50:47] >= 5) shift\_reg[50:47] = shift\_reg[50:47] + 2'b11;
30. **if** (shift\_reg[54:51] >= 5) shift\_reg[54:51] = shift\_reg[54:51] + 2'b11;
31. **if** (shift\_reg[58:55] >= 5) shift\_reg[58:55] = shift\_reg[58:55] + 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**
44. bcd\_code <= shift\_reg[58:27];
45. **end**
46. **end**
47. **end**
49. endmodule

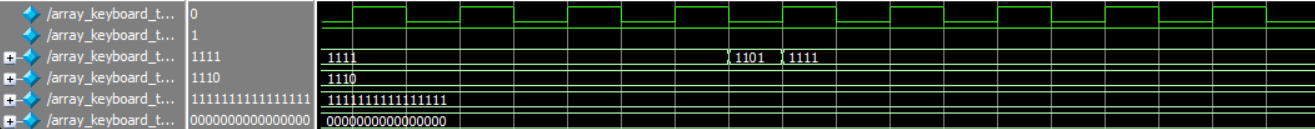
**segment\_scan.v**

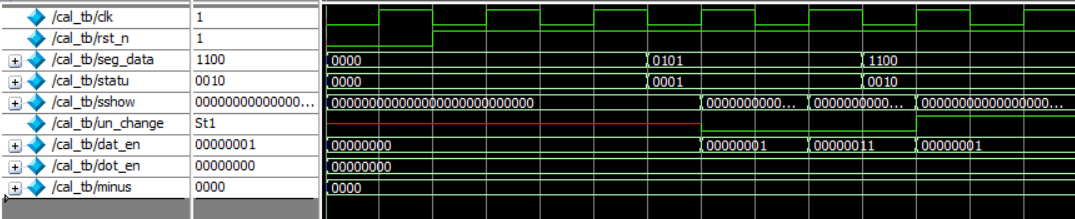
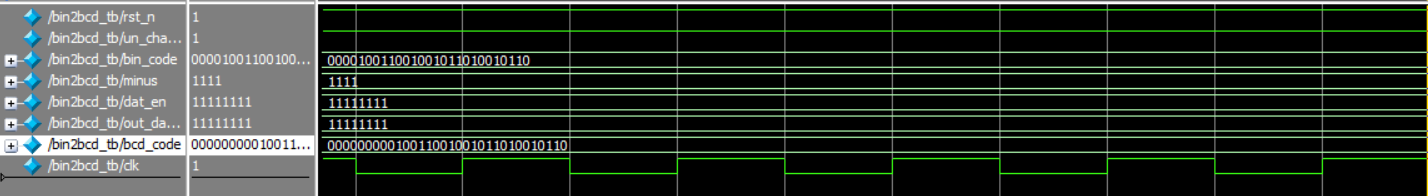
该模块的主要功能是对数码管的数据进行扫描和控制，根据外部输入的数据和使能信号，完成数码管的显示操作。

1. // 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
2. module segment\_scan(
3. input               clk,        //系统时钟 12MHz
4. input               rst\_n,      //系统复位 低有效
5. input       [3:0]   dat\_1,      //SEG1 显示的数据输入
6. input       [3:0]   dat\_2,      //SEG2 显示的数据输入
7. input       [3:0]   dat\_3,      //SEG3 显示的数据输入
8. input       [3:0]   dat\_4,      //SEG4 显示的数据输入
9. input       [3:0]   dat\_5,      //SEG5 显示的数据输入
10. input       [3:0]   dat\_6,      //SEG6 显示的数据输入
11. input       [3:0]   dat\_7,      //SEG7 显示的数据输入
12. input       [3:0]   dat\_8,      //SEG8 显示的数据输入
13. input       [7:0]   dat\_en,     //数码管数据位显示使能
14. input       [7:0]   dot\_en,     //数码管小数点位显示使能
15. output  reg         seg\_rck,    //74HC595的RCK管脚
16. output  reg         seg\_sck,    //74HC595的SCK管脚
17. output  reg         seg\_din     //74HC595的SER管脚
18. );
20. localparam  CNT\_40KHz = 300;    //分频系数
22. localparam  IDLE    =   3'd0;
23. localparam  MAIN    =   3'd1;
24. localparam  WRITE   =   3'd2;
25. localparam  LOW     =   1'b0;
26. 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**
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**
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**
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**
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

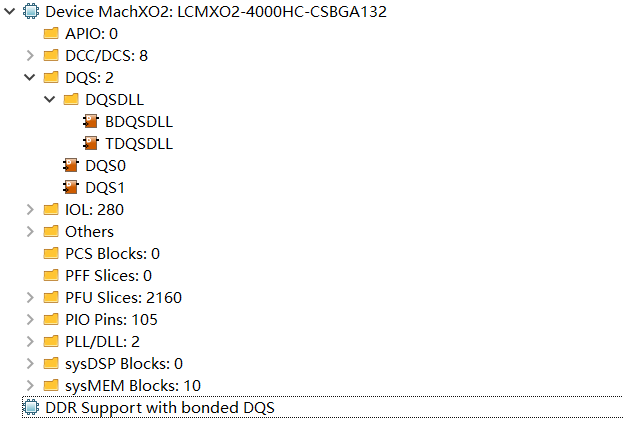
87. **case**(cnt\_main)
88. //对8位数码管逐位扫描
89. //data          [15:8]为段选，         [7:0]为位选
90. 3'd0: data <= {{dot\_en[7],seg[dat\_1]},dat\_en[7]?8'hfe:8'hff};
91. 3'd1: data <= {{dot\_en[6],seg[dat\_2]},dat\_en[6]?8'hfd:8'hff};
92. 3'd2: data <= {{dot\_en[5],seg[dat\_3]},dat\_en[5]?8'hfb:8'hff};
93. 3'd3: data <= {{dot\_en[4],seg[dat\_4]},dat\_en[4]?8'hf7:8'hff};
94. 3'd4: data <= {{dot\_en[3],seg[dat\_5]},dat\_en[3]?8'hef:8'hff};
95. 3'd5: data <= {{dot\_en[2],seg[dat\_6]},dat\_en[2]?8'hdf:8'hff};
96. 3'd6: data <= {{dot\_en[1],seg[dat\_7]},dat\_en[1]?8'hbf:8'hff};
97. 3'd7: data <= {{dot\_en[0],seg[dat\_8]},dat\_en[0]?8'h7f: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**
133. 6'd26: **begin** seg\_sck <= LOW; seg\_din <= data[2]; **end**
134. 6'd27: **begin** seg\_sck <= HIGH; **end**
135. 6'd28: **begin** seg\_sck <= LOW; seg\_din <= data[1]; **end**
136. 6'd29: **begin** seg\_sck <= HIGH; **end**
137. 6'd30: **begin** seg\_sck <= LOW; seg\_din <= data[0]; **end**
138. 6'd31: **begin** seg\_sck <= HIGH; **end**
139. 6'd32: **begin** seg\_rck <= HIGH; **end**                                //当16位数据传送完成后RCK拉高，输出生效
140. 6'd33: **begin** seg\_rck <= LOW; state <= MAIN; **end**
141. default: ;
142. endcase
143. **end**
144. default: state <= IDLE;
145. endcase
146. **end**
147. **end**
149. 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=1551294792&bvid=BV1py421B7JH&cid=1461926353&p=1" scrolling="no" border="0" frameborder="no" framespacing="0" allowfullscreen="true"> </iframe>**