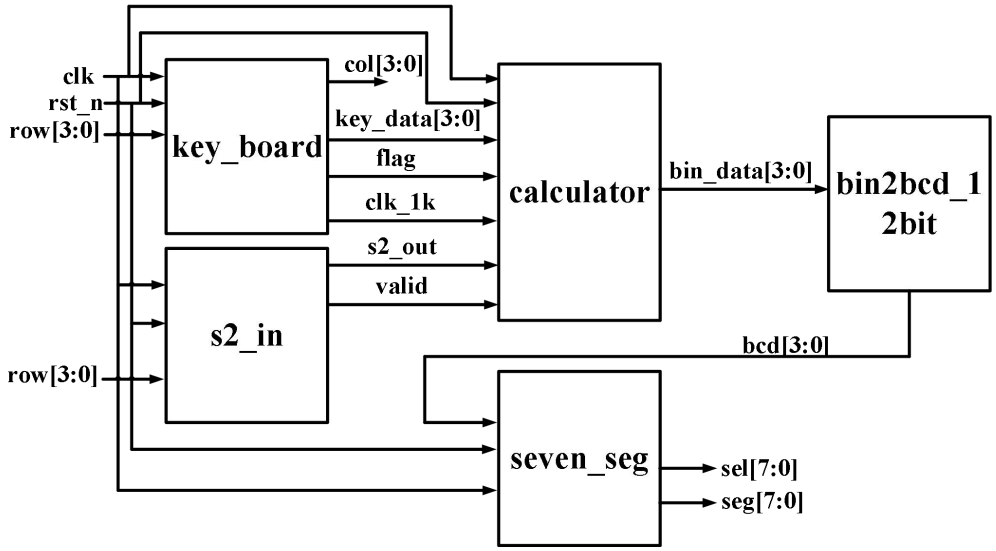
注：本设计报告中各个部分如果页数不够，请大家自行扩页，原则是一定要把报告写详细，能说明设计的成果和特色。报告中应该叙述设计中的每个模块。设计报告将是评定每个人成绩的重要组成部分（**设计内容及报告写作**都作为评分依据）。

设计的功能描述
本次实验设计的是一个十进制正整数计算器，支持十进制整数的加、减、乘、除（求商）4 种运算，且支持连续运算功能； 按键开关 S1 作为异步复位信号，且当 S1 按下时，系统被复位等待数据输入； 按键开关 S2 作为每次计算启动信号（类似计算器的“等于号"功能）； 操作数由 4*4 矩阵键盘输入； 输入数据范围为 2 位十进制正整数； 运算操作由 ABCD 代替，分别代表加、减、乘、除； 连续运算功能举例:先输入 1+2,按下 S2 算出结果为 3，再输入+3,跟上一步算出的结果直接计算，得到结果是 6； 计算结果输出到数码管上，能够下显示输入的数据以及计算的结果，使用六个数码管显示。

系统功能详细设计

用硬件框图描述系统主要功能及各模块之间的相互关系

硬件框图如下：



由 key_board、s2_in、calculator、bin2bcd_12bit 和 seven_seg 五个模块构成，key_board 模块是实现矩阵键盘的输入，s2_in 实现按键 s2 的消抖与输入，calculator 实现计算器中各种运算的实现，bin2bcd_12bit 负责将计算模块的输出数据转化为 bcd 码，seven_seg 模块实现 bcd 码的输出显示。

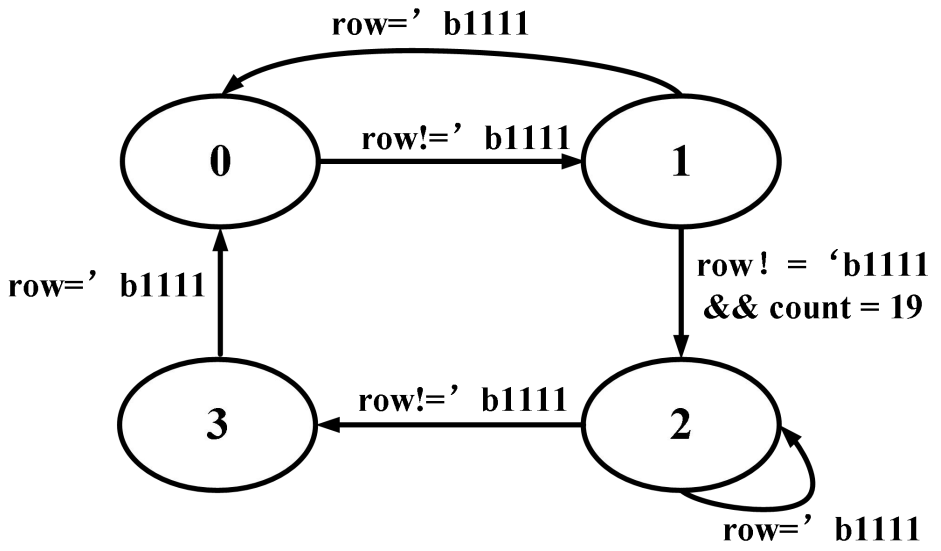
模块设计与实现

包括各子模块功能、设计思路，输入、输出端口及关键代码

1. Keyboard 模块：采用的是 4*4 矩阵键盘电路，矩阵键盘又称行列式键盘，它是用 4 条 I/O 线作为行线，4 条 I/O 线作为列线组成的键盘。在行线和列线的每一个交叉点上，设置一个按键。这样键盘中的按键的个数就是 4*4=16 个。这种行列式键盘结构能够有效地提高单片机系统中 I/O 口的利用率。其中的 ROW0，ROW1，ROW2，ROW3，以及 COL0，COL1，COL2，COL3 信号分别连接到了 FPGA 上，使用一种独特的方法，这 8 个信号能够表示出 16 个按键各自按下的状态。其中输入端口为 clk(时钟信号)、rst（复位信号，按键 S1）、row（键盘扫描行信号);输出端口有：flag（键盘按下使能信号）、keydata（键盘按下具体数字信号）、col（键盘扫描列信号）；

状态描述：0：初始状态；1：检测到有矩阵键盘输入并消抖；2：矩阵键盘列扫描；3：矩阵键盘输出有效

状态转移图：



主要设计代码：

```
module key_board (clk, rst_n, row, col, data, valid, clk_1k);

  input clk;
  input rst_n;
  input [3:0] row;

  output reg [3:0] col;
  output reg [3:0] data;
  output reg valid;
  output reg clk_1k;

  reg [14:0] cnt;
  parameter T1ms = 24999;
  always @ (posedge clk or negedge rst_n)
    begin
      if (rst_n)
        begin
          clk_1k <= 1'b1;
          cnt <= 15'd0;
        end
      else
        begin
          if (cnt < T1ms)
            begin
              cnt <= cnt + 15'd1;
            end
          else
            begin
              cnt <= 15'd0;
              clk_1k <= ~clk_1k;
            end
        end
    end

  reg [7:0] row_col;
  reg [1:0] state;
  reg [4:0] count;
```

```
  always @ (posedge clk_1k or negedge rst_n)
          case (c_state)
            0 :
                if (row !== 4'b1111)
                  n_state <= 1;
            1 : begin
                if (row == 4'b1111)
                    n_state <= 0;
                else if (count >= 19)
                    n_state <= 2;
                end
            2 : begin
                if (row == 4'b1111)
                    n_state <= 2;
                else
                    n_state <= 3;
              end
            3 : begin
                if (row == 4'b1111)
                    n_state <= 0;
                else
                    n_state <= 3;
              end
            default : n_state <= 0;
          endcase
```

```
always @(posedge clk or negedge rst_n) begin
  if(rst_n) begin
    col <= 4'b0000;
    row_col <= 8'd0;
    valid <= 0;
    count <= 0;
  end else begin
    case (c_state)
      1 : begin
            if (row == 4'b1111)
              begin
                count <= 0;
              end
            else
              begin
                if (count < 19)
                  begin
                    count <= count + 1;
                  end
                else
                  begin
                    count <= 0;
                    col <= 4'b0111;
                  end
              end
          end

        2 : begin
            if (row == 4'b1111)
              begin
                col <= {col[2:0],col[3]};
              end
            else
              begin
                row_col <= {row,col};
                valid <= 1;
              end
          end
```

```
            3 : begin
                if (row == 4'b1111)
                  begin
                    n_state <= 0;
                    valid <= 0;
                  end
                else
                  begin
                    valid <= 0;
                  end
              end
```

```
always @ (*)
  begin
    case (row_col)
      8'b0111_0111 : data = 4'h1;
      8'b0111_1011 : data = 4'h2;
      8'b0111_1101 : data = 4'h3;
      8'b0111_1110 : data = 4'ha;

      8'b1011_0111 : data = 4'h4;
      8'b1011_1011 : data = 4'h5;
      8'b1011_1101 : data = 4'h6;
      8'b1011_1110 : data = 4'hb;

      8'b1101_0111 : data = 4'h7;
      8'b1101_1011 : data = 4'h8;
      8'b1101_1101 : data = 4'h9;
      8'b1101_1110 : data = 4'hc;

      8'b1110_0111 : data = 4'he;//*
      8'b1110_1011 : data = 4'h0;
      8'b1110_1101 : data = 4'hf;//#
      8'b1110_1110 : data = 4'hd;

      default : data = 4'h0;
    endcase
  end
```

2. S2_in 模块：为 s2 按键的输入模块，按键输入需要做相应的按键消抖，整体的设计思路就是下降沿出现开始计数 15ms，如果在计数 15ms 中又出现了一个上升沿，则计数清零，等待下一个下降沿，然后接着计数，反复如此，直到在 15ms 内未出现上升沿，算是真正的稳定。其中输入端口为：clk(时钟信号)、rst（复位信号，按键 S1）、s2_in（按键输入信号）；输出端口有：valid（按键有效信号）、s2_out（按键输出信号）

主要设计代码：

```
module s2_in(
    input wire clk,
    input wire rst_n,
    input wire s2_in,
    output reg valid,
    output reg s2_out
);

    reg cnt_inc = 1'b0;//增加信号
    reg [24:0] cnt;//底层计数器?
    wire cnt_end;
    assign cnt_end = cnt_inc & (cnt == 25'd30);//
    always @(posedge clk or posedge rst_n)begin
        if (rst_n)
            cnt_inc <= 1'b0;
        else
            cnt_inc <= 1'b1;
    end
    always @(posedge clk or posedge rst_n)begin
        if (rst_n)
            cnt  <= 25'd0;
        else if (cnt_end )
            cnt  <= 25'd0;
        else if (cnt_inc)
            cnt  <= cnt  + 25'd1;
    end
    reg sig_r0 = 1'b0;
    reg sig_r1 = 1'b0;
    always @(posedge clk or posedge rst_n)begin
        if(rst_n)
            sig_r0 <= 1'b0;
        else if(cnt_end)
            sig_r0 <= s2_in;
        else
            sig_r0 <= sig_r0;
    end
```

```
    always @(posedge clk or posedge rst_n)begin
        if(rst_n)
            sig_r1 <= 1'b0;
        else
            sig_r1 <= sig_r0;
    end
    always @(posedge clk or posedge rst_n)begin
        if(rst_n)
            sig_r1 <= 1'b0;
        else
            sig_r1 <= sig_r0;
    end
    always @(posedge clk or posedge rst_n)begin
        if(rst_n)
            s2_out <= 1'b0;
        else if(~sig_r1 & sig_r0)begin
            s2_out <= 1;
            valid <= 1;
        end
        else begin
            s2_out <= 0;
            valid <= 0;
        end
    end
endmodule
```

3. Calculator 模块：存储部分用状态机和寄存器来实现，我们输入的数字应用了移位拼接的原理，若第一个输入的是数字键，则保存下来，第二次输入还是输入的是数字键时，第一个数值左移变为十位，第二次的为个位，第三次若还是数字键，那么第一个数值将变为百位，第二个为十位，第三个为个位，以此类推，直到有符号键输入。

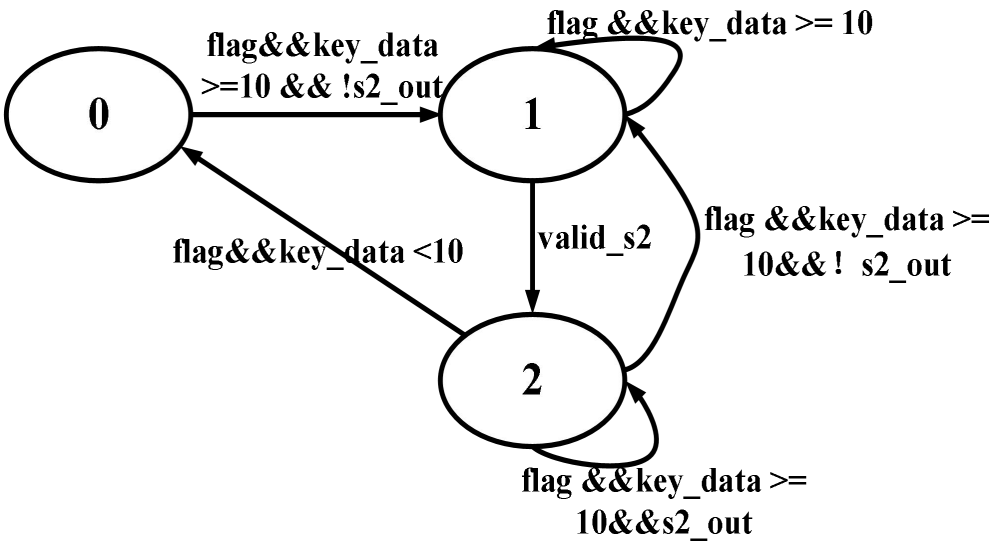
当进行第一次计算时，第一个数字存放在 num1 里面。按下运算符以后，第二个数字放在 bin_data 里面。当再按下运算符号或者等号时，第一次计算的结果将存放在 ans 里面，同时 reg 清零，等待下一个数字的输入。进行第二次运算时，将 num1 里面的结果与 reg 里面新输入的数字进行运算，再将运算结果存放在 num1 里面，直到最后

按下等号按键的时候，显示最终的运算结果。

输入端口为：clk(时钟信号)、rst（复位信号，按键 S1）、row（键盘扫描行信号）、 flag（键盘按下使能信号）、keydata（键盘按下具体数字信号）、s2_out（按键输出信号）、valid_s2（按键输出有效信号），输出端口有：bin_data（计算结果信号）

状态描述：0：初始状态，存储矩阵键盘的输入，在接收到矩阵键盘输入为运算符号时跳转；1：计算状态，检测到运算符的输入后将数据暂存，与此时矩阵键盘的输入进行运算并跳转；2：在输入为运算符时可连续计算，为数字时则进行下次计算

状态转移图：



主要设计代码：

```
module calculator (clk, rst_n, flag, key_data, s2_out, valid_s2,bin_data);

  input clk;
  input rst_n;
  input flag;
  input [3:0] key_data;
  input s2_out;
  input valid_s2;

  output reg [19:0] bin_data;

  reg [1:0] c_state;
  reg [1:0] n_state;
  reg [19:0] num1;
  reg [3:0] opcode;

  always @(posedge clk or negedge rst_n) begin
    if(rst_n) begin
      c_state <= 0;
    end else begin
      c_state <= n_state;
    end
  end
always @ (*)
    begin
        begin
          case (c_state)
            0 : begin
                if (flag)
                  begin
                    if (key_data >= 10)
                      begin
                        if (s2_out)
                          begin
                            n_state = 0;
                          end
                        else
                          begin
                            n_state = 1;
                          end
                      end
                  end
```

```
            1 : begin
                if (flag)
                  begin
                    if (key_data >= 10)
                      begin
                        n_state = 1;
                      end
                  end
                else if (valid_s2) begin
                  n_state = 2;
                  end
                else
                  begin
                    n_state = 1;
                  end
              end

            2 : begin
                if (flag)
                  begin
                    if (key_data < 10)
                      begin
                        n_state = 0;
                      end
                    else
                      begin
                        if (s2_out)
                          begin
                            n_state = 2;
                          end
                        else
                          begin
                            n_state =1;
                          end
                      end
                  end
                else
                  begin
                    n_state = 2;
                  end
              end
          endcase
```

```
always @(posedge clk or negedge rst_n) begin
    begin
      if (rst_n)
        begin
          num1 <= 0;
          bin_data <= 0;
          opcode <= 0;
        end
      else
        begin
          case (c_state)
            0 : begin
                if (flag)
                  begin
                    if (key_data < 10)
                      begin
                        bin_data <= bin_data * 10 + key_data;
                      end
                    else
                      begin
                        if (!s2_out)
                          begin
                            opcode <= key_data;
                            num1 <= bin_data;
                            bin_data <= 0;
                          end
                      end
                  end
              end
```

```
            1 : begin
                if (flag)
                  begin
                    if (key_data < 10)
                      begin
                        bin_data <= bin_data * 10 + key_data;
                      end
                  end
                else if (valid_s2) begin
                    case (opcode)
                      10 :  begin bin_data <= num1 + bin_data;end
                      11 :  begin bin_data <= num1 - bin_data;end
                      12 :  begin bin_data <= num1 * bin_data;end
                      13 :   begin bin_data <= num1 / bin_data;end
                      default : bin_data <= 0;
                    endcase
                  end
              end
```

```
            2 : begin
                if (flag)
                  begin
                    if (key_data < 10)
                      begin
                        bin_data <= {16'd0,key_data};
                      end
                    else
                      begin
                        if (!s2_out)
                          begin
                            num1 <= bin_data;
                            opcode <= key_data;
                            bin_data <= 0;
                          end
                      end
                  end
              end
          endcase
```

4. Bcd 模块：由于最终要显示十进制的数据，所以要把二进制的数据做 bcd 码的转换，继而通过数码管显示，转换过程通过除法与取余实现。输入端口为: bin（计算模块输出数据信号），输出端口为 bcd（转换为 bcd 码后的数据信号）

主要设计代码：

```
module bin2bcd_12bit(bin, bcd);

  input [19:0] bin;
  output reg [23:0] bcd;

  always @ (*)
    begin
      bcd[3:0] = bin%10;
      bcd[7:4] = bin/10%10;
      bcd[11:8] = bin/100%10;
      bcd [15:12] = bin/1000%10;
      bcd[19:16] = bin/10000%10;
      bcd[23:20] = bin/100000%10;
    end

endmodule
```

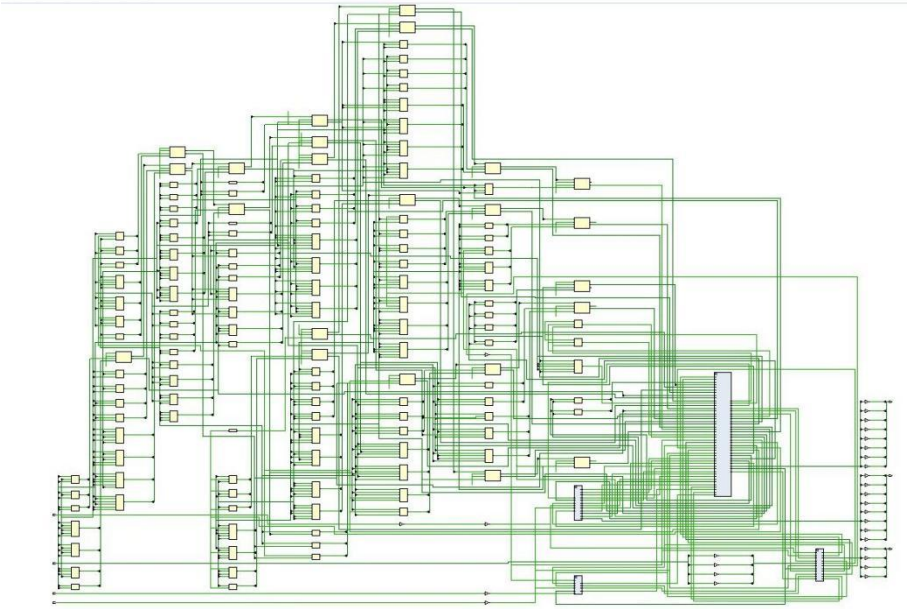
5. seven_seg 模块：显示部分是系统的输出部分，用于显示按键值及计算结果，由于数字系统的数据运算都是二进制的，而输出表达式都是 BCD 码，为了满足 BCD 码的译码显示，最方便的方法就是利用译码程序在 FPGA 中实现。本文采用的是共阳极 7 段数码管，显示数字时需要将对应管脚置为低电平。输入端口为 clk(时钟信号)、rst（复位信号，按键 S1）、data_in（转换后的 bcd 数据信号）;输出端口有：sel（数码管位选信号）、seg（数码管片选信号）;

主要设计代码：

```
always @ (*)
  begin
    if (rst_n)
      begin
        seg = 8'b0000_0000;
      end
    else
      begin
        case (num)
          0 : seg = 8'b1100_0000;
          1 : seg = 8'b1111_1001;
          2 : seg = 8'b1010_0100;
          3 : seg = 8'b1011_0000;
          4 : seg = 8'b1001_1001;
          5 : seg = 8'b1001_0010;
          6 : seg = 8'b1000_0010;
          7 : seg = 8'b1111_1000;
          8 : seg = 8'b1000_0000;
          9 : seg = 8'b1001_0000;
          default : seg = 8'b0000_0000;
        endcase
      end
  end
```

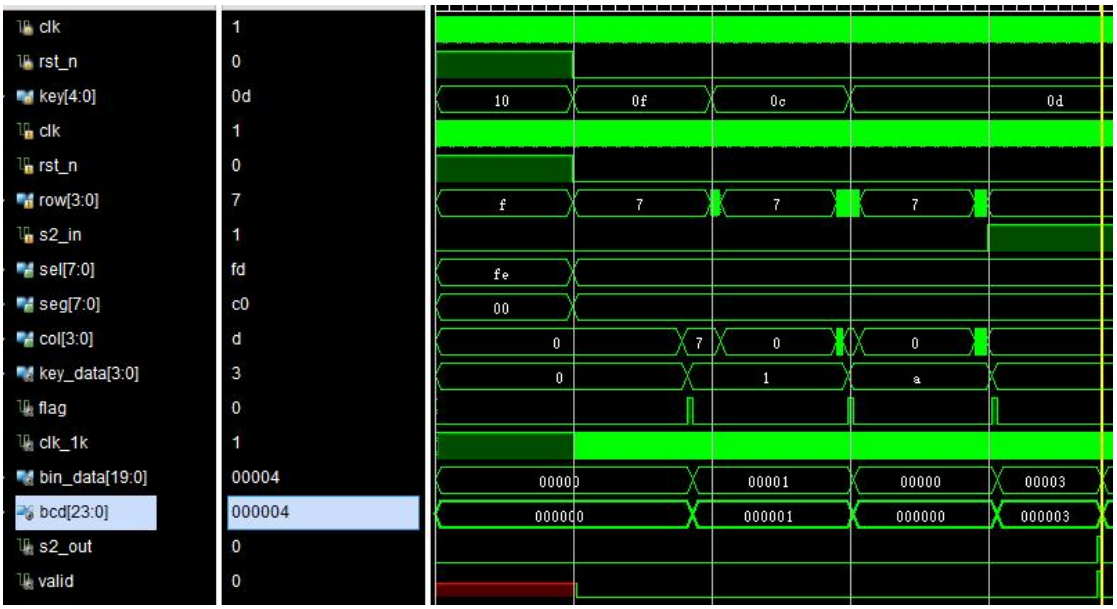
```
always @ (posedge clk or negedge rst_n)
  begin
    if (rst_n)
      begin
        sel <= 8'b1111_1110;
      end
    else
      begin
        sel <= {sel[6:0],sel[7]};
      end
  end

always@(*)
  begin
    if (rst_n)
      begin
        num = 0;
      end
    else
      begin
        case (sel)
          8'b1111_1110 : num = data_in[23:20];
          8'b1111_1101 : num = data_in[19:16];
          8'b1111_1011 : num = data_in[15:12];
          8'b1111_0111 : num = data_in[11:8];
          8'b1110_1111 : num = data_in[7:4];
          8'b1101_1111 : num = data_in[3:0];
          default : num = 0;
        endcase
      end
  end
```

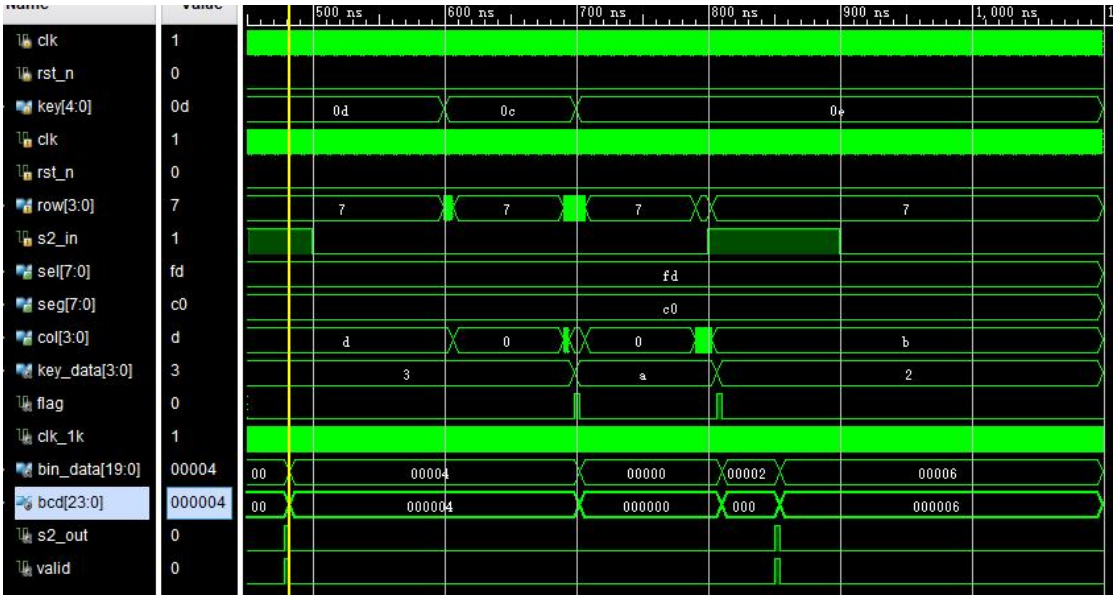
6. 顶层模块的 RTL 分析原理图截图



调试报告

仿真波形截图及仿真分析





波形分析：

Top 模块完成计算模块的输入，进行 1+2+3 连续计算功能的仿真，输入时钟信号 clk、复位信号 rst_n、按键信号 s2，设置内部分频处所用到的参数 T1ms，键盘扫描行信号 row，通过 key 与阵列转换信号来进行按键的输入；输出信号有数码管的位选信号 sel、数码管的段选

信号 seg、按键输出列信号 col。从上述波形可以看出：

（0）时钟信号以 1ns 为周期；

（1）初始态输入复位下信号为 1，按键 key 信号为'h10，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b1111;矩阵按键输出信号 keydata 为 0，计算模块输出信号 bin_data 为 0，bcd 转换模块输出信号 bcd 为 0，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4'b1111，数码管位选信号 sel 为'hfe，数码管段选信号 seg 为'h00，符合预期；

（2）100ns 时输入复位下信号为 0，按键 key 信号为'hfd，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b111; 矩阵按键输出信号 keydata 为 0，计算模块输出信号 bin_data 为 0，bcd 转换模块输出信号 bcd 为 0，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4'b0，数码管位选信号 sel 为'hfd，数码管段选信号 seg 为'hc0，符合预期；

（3）186.5ns 时输入复位下信号为 0，按键 key 信号为'h0f，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b1111; 矩阵按键输出信号 keydata 为 1，计算模块输出信号 bin_data 为 1，bcd 转换模块输出信号 bcd 为 1，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 1，输出键盘扫描列信号 col 为 4'b111，数码管位选信号 sel 为'hfd，数码管段选信号 seg 为'hc0，符合预期；

（4）300ns 时输入复位下信号为 0，按键 key 信号为'h0c，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b111，矩阵按键输出信号

keydata 为 a，计算模块输出信号 bin_data 为 0，bcd 转换模块输出信号 bcd 为 0，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 1，输出键盘扫描列信号 col 为 4'b1111，数码管位选信号 sel 为'hfd，数码管段选信号 seg 为'hc0，符合预期；

（5）406.5ns 时输入复位下信号为 0，按键 key 信号为'h0d，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b111;计算模块输出信号 bin_data 为 3，bcd 转换模块输出信号 bcd 为 3，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 1，输出键盘扫描列信号 col 为 4'b1111,数码管位选信号 sel 为'hfe，数码管段选信号 seg 为'hc0，符合预期；

（6）478.5ns 时输入复位下信号为 0，按键 key 信号为'h0d，s2_in 按键输入信号为 1，键盘扫描行信号 row 为 4'b111;计算模块输出信号 bin_data 为 3，bcd 转换模块输出信号 bcd 为 3，按键输出信号 s2_out 为 1，按键输出有效信号 valid 为 1，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4'b1101,数码管位选信号 sel 为'hfb，数码管段选信号 seg 为'hc0，符合预期；

（7）600ns 时输入复位下信号为 1，按键 key 信号为'h0d，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4'b111;计算模块输出信号 bin_data 为 4，bcd 转换模块输出信号 bcd 为 4，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4'b1111,数码管位选信号 sel 为'hf7，数码管段选信号 seg 为'hc0，符合预期；

（8）700ns 时输入复位下信号为 0，按键 key 信号为’h0c，s2_in 按键输入信号为 0，键盘扫描行信号 row 为 4’b1111;计算模块输出信号 bin_data 为 0，bcd 转换模块输出信号 bcd 为 0，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 1，输出键盘扫描列信号 col 为 4’b1110,数码管位选信号 sel 为’hef，数码管段选信号 seg 为’hc0，符合预期；

（9）800ns 时输入复位下信号为 0，按键 key 信号为’h0e，s2_in 按键输入信号为 1，键盘扫描行信号 row 为 4’b111;计算模块输出信号 bin_data 为 0，bcd 转换模块输出信号 bcd 为 0，按键输出信号 s2_out 为 0，按键输出有效信号 valid 为 0，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4’b1101,数码管位选信号 sel 为’hef，数码管段选信号 seg 为’hc0，符合预期；

（10）854.5ns 时输入复位下信号为 0，按键 key 信号为’hb，s2_in 按键输入信号为 1，键盘扫描行信号 row 为 4’b1111;计算模块输出信号 bin_data 为 6，bcd 转换模块输出信号 bcd 为，按键输出信号 s2_out 为 1，按键输出有效信号 valid 为 1，矩阵键盘输出有效信号 flag 为 0，输出键盘扫描列信号 col 为 4’b1011,数码管位选信号 sel 为’hef，数码管段选信号 seg 为’h82，符合预期；

故根据上述分析，模块实现了 1+2+3 连续计算。

设计过程中遇到的问题及解决方法
遇到的问题有： 1. 对动态数码管的位选信号理解不到位，因为是共阳极数码管，低电平显示，刚开始使用的移位位选信号的方法错误。解决方法：使用循环移位的方法进行位选信号的移位； 2. 在 s2 的按键输入过程中，按键输入的逻辑与计算模块本身的逻辑产生了冲突,导致计算模块一直在进行连续地计算 ans 与运算符相运算的结果，导致了计算模块的计算结果显示不对，逻辑出现错误。解决方法：修改逻辑，与矩阵按键本身的逻辑相结合后进行输出，结果显示正确； 3. 按键 s2 按下后没有反应。解决方法：按键消抖过程需要进行一定时间的计时，比如 15ms，在设计中由于使用的是上层的分频时钟，导致这个时间增大了 1k 倍故按下 s2 后无反应，且按键消抖后需要检测上升沿进行输出。

课程设计总结
包括设计的总结和还需改进的内容以及收获 总结与收获： 我在数字逻辑设计实验中培养了时序逻辑的思维，锻炼了独立设计、分析电路的能力，能够独当一面；此外，还充分意识到了自顶向下的设计意识的重要性——直接上手打代码很容易导致设计的逻辑混乱，混淆模块之间的功能和信号的传递；同时，也充分感受到了模块的灵活使用对于成功完成实验的意义——当某个模块代码量过大时，使用子模块将同一功能的语句封装在一起，专注于宏观功能而不涉及底层实现，在该模块没有问题的情况下可以极大减少代码阅读量，同时更便于 debug 时更快更精准地定位到出错的位置；这门课也大大提升了我的抗压能力，在心态上也算是经历了一场磨炼，每一次被未知的 bug 折磨到濒临崩溃之后，就像破茧成蝶，总有新的成长、新的蜕变。 需要改进的地方： 理论课和实验课可以衔接得更紧密些；可以增设几节课专门用于实验的答疑课，或者是建立共享文档集中解答疑惑。