数字逻辑Project报告

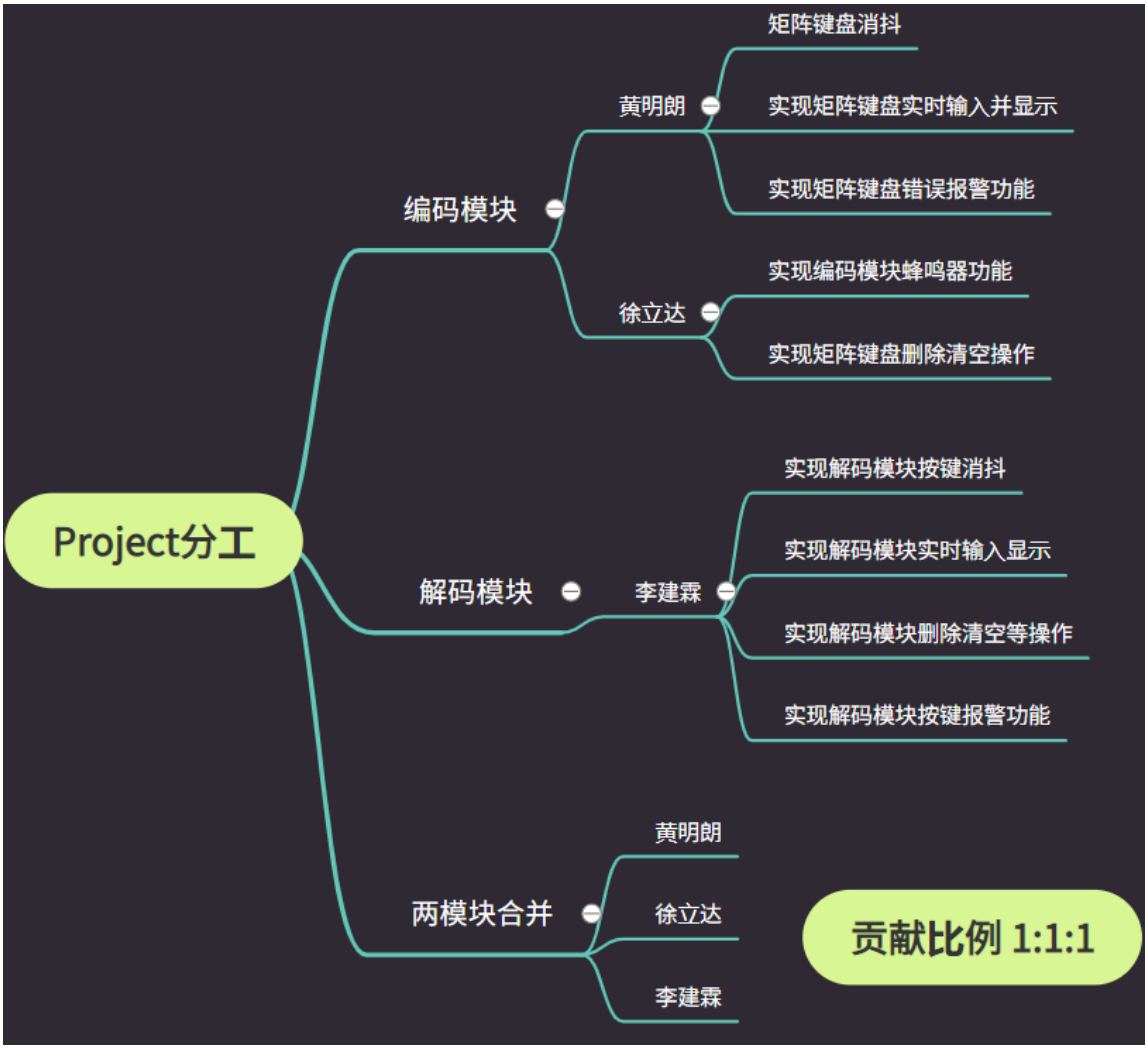
小组成员：李建霖(12012221)、黄明朗(12011828)、徐立达(12010624)

开发计划

- 小组选题

  $FPGA$ 摩斯电码

▼ 成员分工以及贡献百分比



▼ 进度安排以及执行记录

▼ Week 12

- [x] ~~讨论确定project选题~~
- [x] ~~制定接下来几周的project执行计划~~

▼ Week 13

- [x] ~~熟悉project要求~~
- [x] ~~设计project大致的状态流程图~~
- [x] ~~进行project大致分工~~

▼ Week 14

- [x] ~~确立顶层模块输入、输出端口~~
- [x] ~~完成相应模块输入输出端口设计~~
- [x] ~~完成相应模块的端口连接~~

▼ Week 15

- 编码模块部分：
- [x] ~~实现矩阵键盘按键消抖~~
- [x] ~~实现矩阵键盘的连续输入并连续显示~~
- [x] ~~实现矩阵键盘的清空、Backspace功能~~
- 解码模块部分：
- [x] ~~实现解码模块所需按键的消抖~~
- [x] ~~实现长短码的LED 显示~~

▼ Week 16

- 编码模块部分：
- [x] ~~实现编码模块蜂鸣器播放功能~~
- [x] ~~实现编码模块输入溢出的错误报警功能~~
- 解码模块部分：
- [x] ~~利用按键实现对长短码的删除、清空功能~~
- [x] ~~实现输入异常，解码异常等报警功能~~

- 顶层模块部分：

- [x] ~~用状态转换器实现解码、编码状态的切换~~
- [x] ~~实现编码、解码模块的上板测试~~
- [x] ~~实现顶层模块的上板测试~~

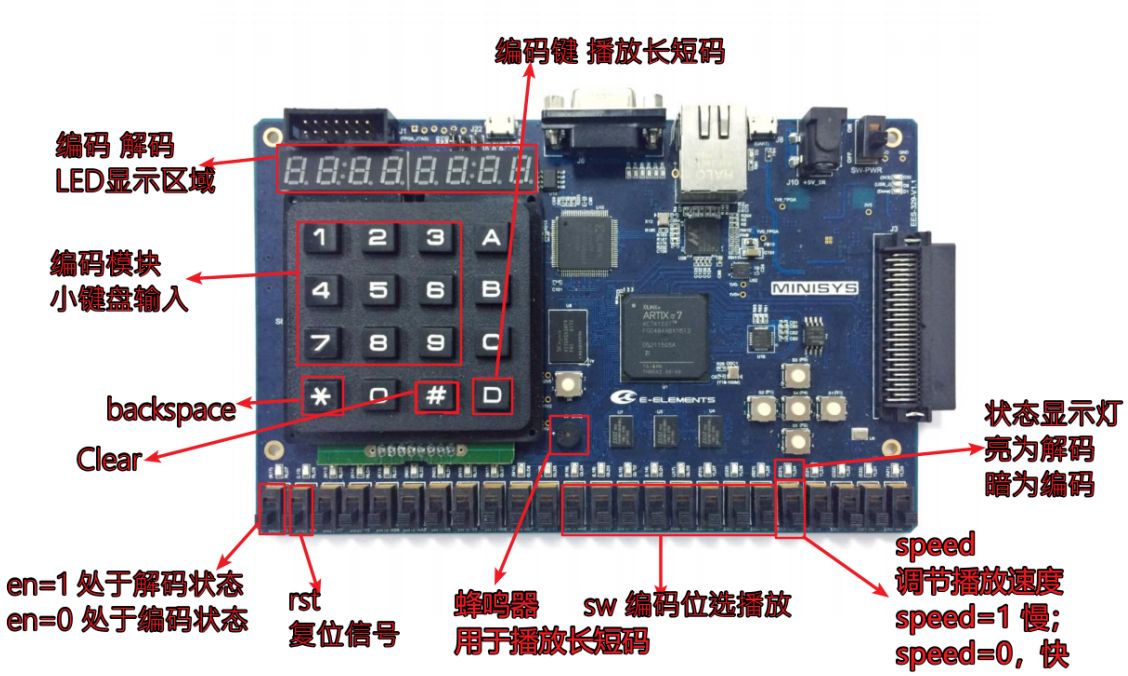
- project其他部分

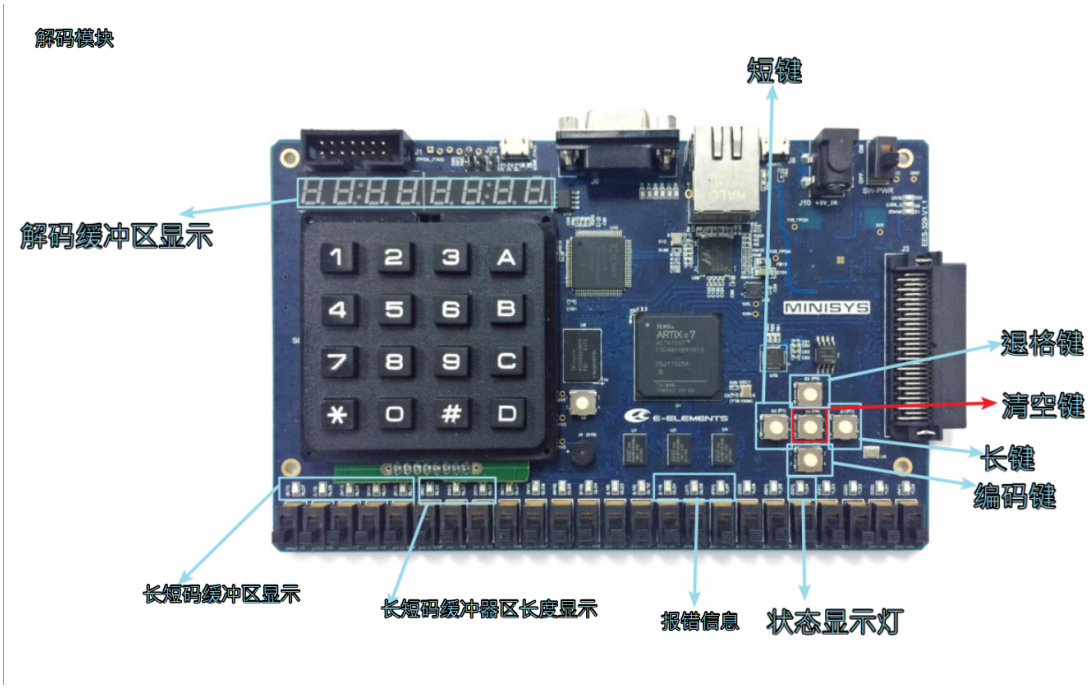
- [x] ~~录制测试视频~~
- [x] ~~编写Project报告~~
- [x] ~~绘制测试手册，端口分布图~~

设计

- 需求分析

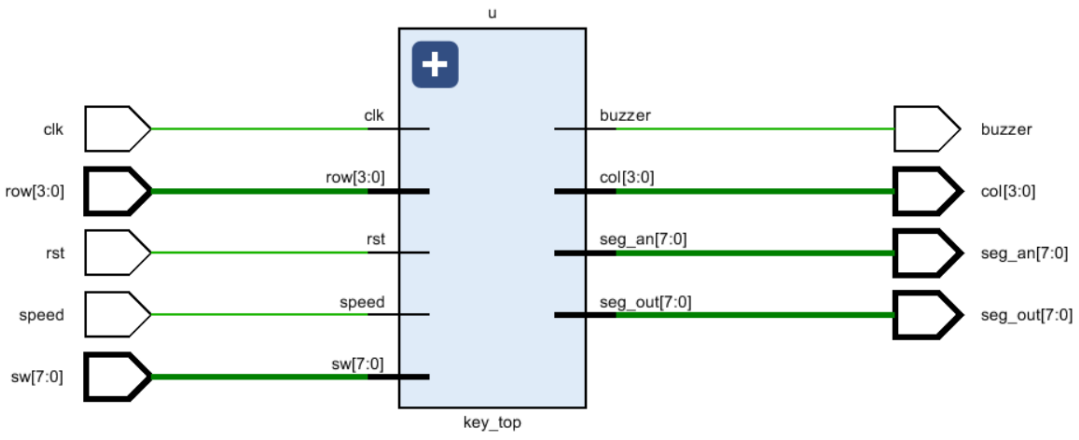
▼ 开发板上使用到的输入输出设备





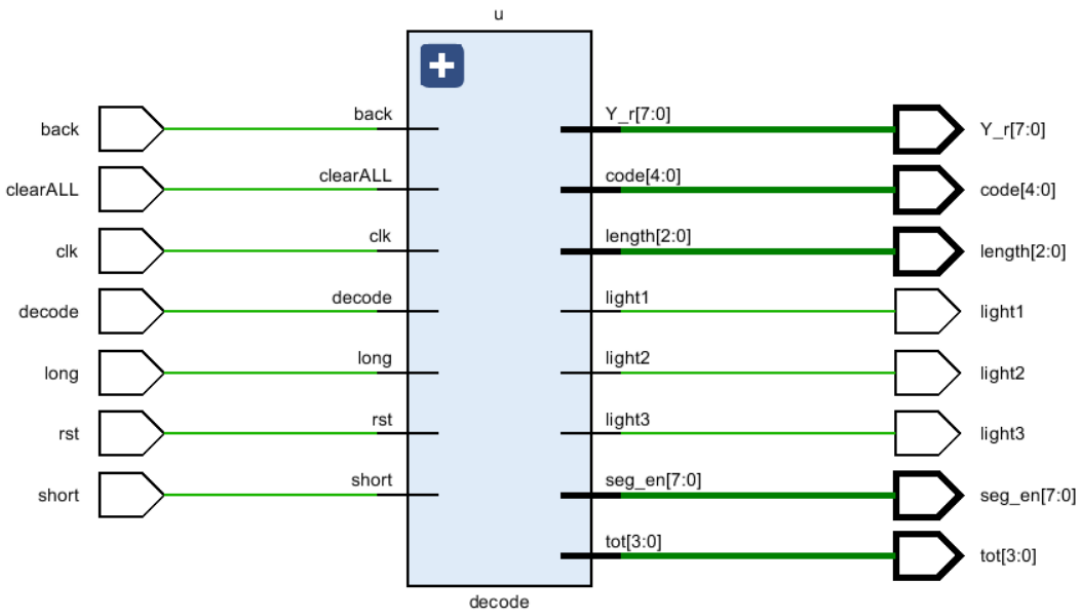
- 系统结构与设计
  - ▼ 各模块接口及功能

    该Project有两个子模块构成
    - ▼ 编码模块



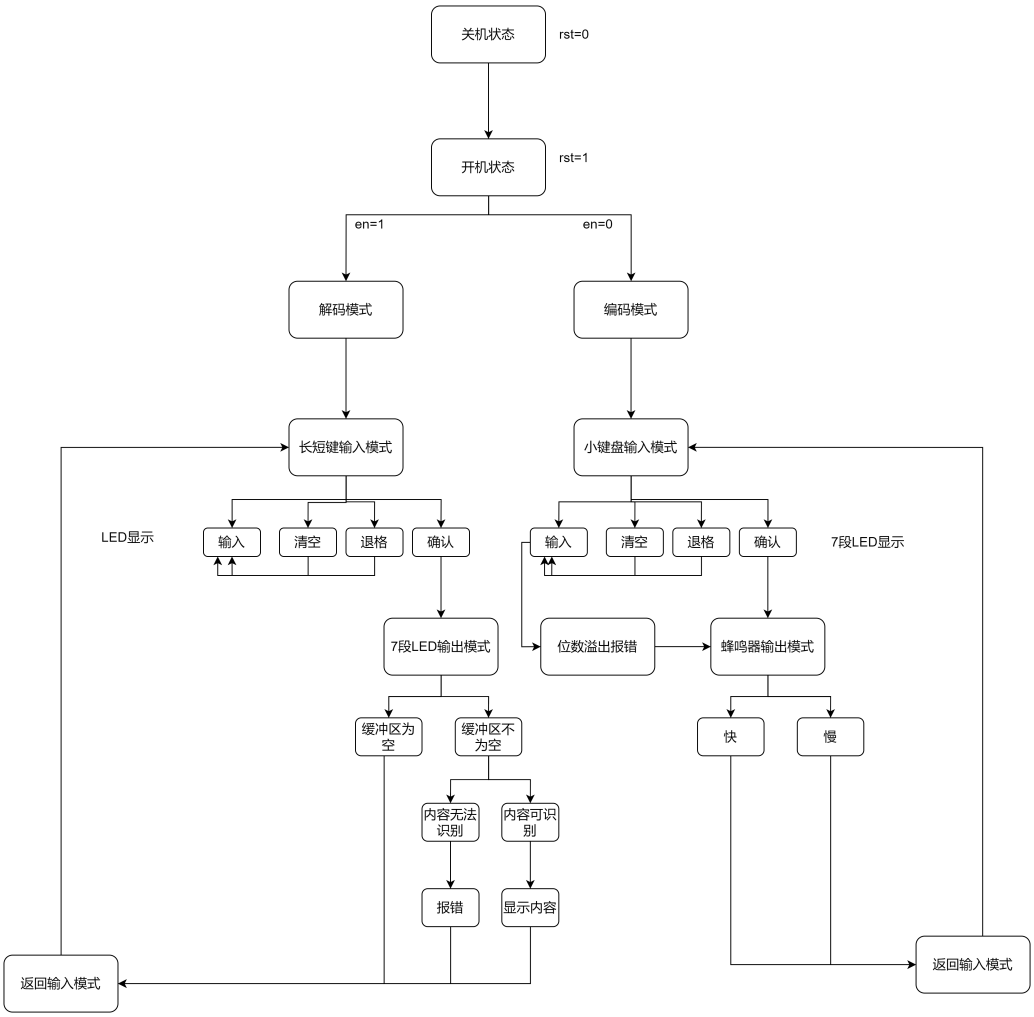
clk—时钟信号	rst—-同步复位信号	buzzer—-蜂鸣器信号	
col—-行信号	row—-行信号	seg_ans—-数码管使能信号	
speed—-速度调节信号	sw—-编码播放位选信号	seg_out—-数码管信号	

▼ 解码模块



clk—时钟信号	rst—-同步复位信号	Y_r—-数码管信号	light1-3—三个报错信号灯
clearALL—清空信号	decode—-解码确认信号	code—-长短码缓冲区显示信号	tot—显示解码总个数
long—长键	short—-短键	length—-长短码个数信号	seg_en—数码管使能信号

▼ 状态迁移\事务处理流程图



- 详细设计

  ▼ 编码模块

  - 分频部分

    分频时钟，将100m时钟分频成1khz，用于矩阵按键和数码管动态显示

```
reg [15:0] cnt; // 分频计数器
reg clk1k_reg;
wire clk1k;
//分频计数器，计数到49999回0
always @ (posedge clk or posedge rst)
  if (rst)
    cnt <= 0;
  else
  if(cnt==49999)
    cnt <= 0;
```

```
  else
    cnt <= cnt + 1'b1;
//clk1k,1k时钟，占空比为50%，计数到49999翻转，100m/((49999+1)*2)=1k
always @ (posedge clk or posedge rst)
  if (rst)
    clk1k_reg <= 0;
  else
    if(cnt==49999)
      clk1k_reg <= ~clk1k_reg;

assign clk1k=clk1k_reg;
```

- 对key_pressed_flag延迟一拍操作

```
//将key_pressed_flag延迟一拍，用于检测上升沿
//形成一个key_pressed_flag_plus_dly1的脉冲信号
always @ (posedge clk1k or posedge rst)
  if (rst)
  begin
    key_pressed_flag_dly1 <= 0;
    key_pressed_flag_dly2 <= 0;
  end
  else
  begin
    key_pressed_flag_dly1 <= key_pressed_flag;
    key_pressed_flag_dly2 <= key_pressed_flag_dly1;
  end

//key_pressed_flag_plus为key_pressed_flag的上升沿，
//即key_pressed_flag=1，key_pressed_flag_dly1=0
assign key_pressed_flag_plus=key_pressed_flag & (~key_pressed_flag_dly1);

//key_pressed_flag_plus_dly1为key_pressed_flag_plus的下一拍，
//即key_pressed_flag_dly1=1，key_pressed_flag_dly2=0
assign key_pressed_flag_plus_dly1
=key_pressed_flag_dly1 & (~key_pressed_flag_dly2);
```

- 蜂鸣器模块

```
//buzzer,在buzzer_en为1时，以500hz翻转，输出蜂鸣器
always @ (posedge clk1k or posedge rst)
  if (rst)
    buzzer <= 0;
  else
    if (buzzer_en)
    buzzer <= ~ buzzer;
```

```
  else
    buzzer <= 0;
endmodule

//然后通过buzzer_en对蜂鸣器进行控制
//倒数buzzer_en_cnt直到0，并对buzzer_en_cnt分范围进行控制
//buzzer_en_cnt,在溢出时，或者按下D时，计数为2000,2s钟，否则每个clk-1，减到0不再减
always @ (posedge clk1k or posedge rst)
  if (rst)begin
    buzzer_en_cnt <= 0;
    //buzzer_en<=0;
  end
  else
    if (key_pressed_flag_plus_dly1)begin
          if (keyboard_val==4'hD)begin
                buzzer_en_cnt <= total;
          end
          else if (keyboard_val<4'hD && key_in_cnt==8)begin
                  //位数溢出后报警
                buzzer_en_cnt <= 2000;
          end
          else begin
                buzzer_en_cnt <= buzzer_en_cnt;
          end
    end
  else if (buzzer_en_cnt!=0)begin
      buzzer_en_cnt <= buzzer_en_cnt-1;
  end

//调节播放速度
assign long =(speed)?14'd2000:14'd1000;
assign short=(speed)?14'd1000:14'd500;
assign mid=(speed)?14'd1000:14'd500;

//对buzzer_en进行分范围控制
case(tmp_seg_data)
     5'd0:begin
         if(buzzer_en_cnt>=total-long)
             buzzer_en=1'b1;
         else if(buzzer_en_cnt>=total-long -mid)
             buzzer_en=1'b0;
         else if(buzzer_en_cnt>=total-long-mid-long)
             buzzer_en=1'b1;
         else if(buzzer_en_cnt>=total-long-mid-long-mid)
             buzzer_en=1'b0;
         else if(buzzer_en_cnt>=total-long-mid-long-mid-long)
             buzzer_en=1'b1;
         else if(buzzer_en_cnt>=total-long-mid-long-mid-long-mid)
             buzzer_en=1'b0;
         else if(buzzer_en_cnt>=total-long-mid-long-mid-long-mid-long)
             buzzer_en=1'b1;
         else if(buzzer_en_cnt>=total-long-mid-long-mid-long-mid-long-mid)
             buzzer_en=1'b0;
         else if(buzzer_en_cnt>=total-long-mid-long-mid-long-mid-long-mid-long)
```

```
            buzzer_en=1'b1;
        else
            buzzer_en=1'b0;
    end
```

- 对按键的次数，即输入位数 进行记录

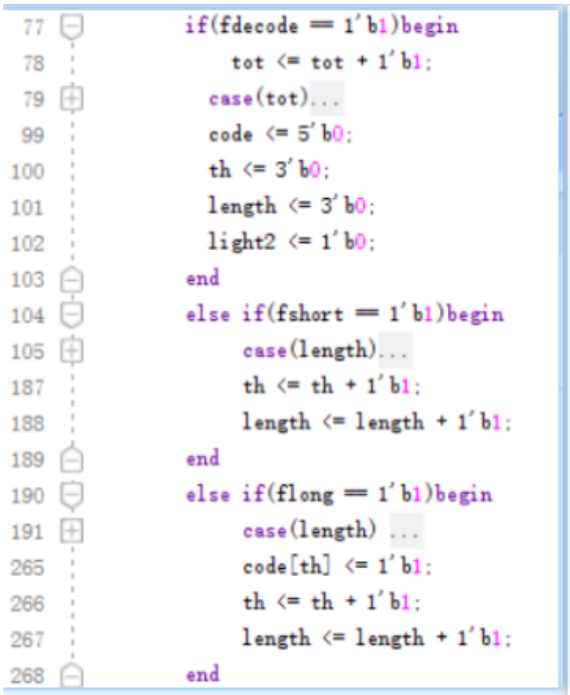
```
//按键次数，每按键1次0-C加1，按键星号-1，按键#号清0，按键D回0
always @ (posedge clk1k or posedge rst)
  if (rst)
    key_in_cnt <= 0;
  else
    if (key_pressed_flag_plus_dly1)
      if (keyboard_val==4'hF || keyboard_val==4'hD)
      key_in_cnt <= 0;
    else if(keyboard_val==4'hE)//按下删除键
      if (key_in_cnt!=0)
        key_in_cnt <= key_in_cnt-1;
      else//
        key_in_cnt <= key_in_cnt;
    else
      if (key_in_cnt!=8)//位数溢出的情况
        key_in_cnt <= key_in_cnt+1;
```

- 按键检测

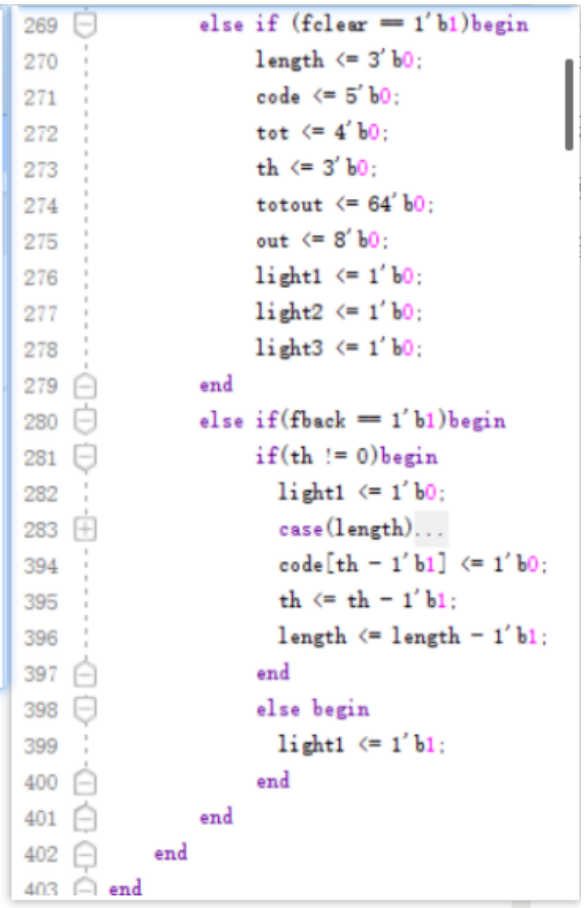
```
//逐个对数码管0，1，2，3，4，5，6，7值进行非阻塞赋值
//数码管1值，在每按键1次0-C时显示数码管0值，按键星号显示数码管2值，
//按键#号清0不显示，按键D不变，再次按键0-C时,不显示
always @ (posedge clk1k or posedge rst)
  if (rst)
    seg_data1 <= 5'h1f;
  else
    if (key_pressed_flag_plus_dly1)//检测到有按键按下
      if (keyboard_val==4'hF)
        seg_data1 <= 5'h1f;
    else if (keyboard_val==4'hE)
      seg_data1 <= seg_data2;
    else if (keyboard_val==4'hD)
      seg_data1 <= seg_data1;
    else
      if(key_in_cnt==0)
        seg_data1 <= 5'h1f;
      else if(key_in_cnt!=8)
        seg_data1 <= seg_data0;
```

▼ 解码模块

- 代码核心骨干



```
77          if(fdecode == 1'b1)begin
78              tot <= tot + 1'b1;
79            case(tot)...
99            code <= 5'b0;
100           th <= 3'b0;
101           length <= 3'b0;
102           light2 <= 1'b0;
103         end
104         else if(fshort == 1'b1)begin
105           case(length)...
187           th <= th + 1'b1;
188           length <= length + 1'b1;
189         end
190         else if(flong == 1'b1)begin
191           case(length) ...
265           code[th] <= 1'b1;
266           th <= th + 1'b1;
267           length <= length + 1'b1;
268         end
```



```
269         else if (fclear == 1'b1)begin
270             length <= 3'b0;
271             code <= 5'b0;
272             tot <= 4'b0;
273             th <= 3'b0;
274             totout <= 64'b0;
275             out <= 8'b0;
276             light1 <= 1'b0;
277             light2 <= 1'b0;
278             light3 <= 1'b0;
279         end
280         else if(fback == 1'b1)begin
281             if(th != 0)begin
282               light1 <= 1'b0;
283               case(length)...
394               code[th - 1'b1] <= 1'b0;
395               th <= th - 1'b1;
396               length <= length - 1'b1;
397             end
398             else begin
399               light1 <= 1'b1;
400             end
401         end
402     end
403 end
```

```
always @(posedge clk, posedge rst)
begin
if(en == 1'b1) begin //当en信号为高电平时进入解码模块
    if(rst)begin  //初始化
        tot <= 4'b0;//解码总个数
        code <= 5'b0;//长短码缓冲区
        th <= 3'b0;//警报显示灯
        length <= 3'b0;//长短码缓冲区长度显示
        totout <= 64'b0;//数码管内容存储
    end
    else if(fdecode == 1'b1)begin
        //当decode按键按下，decode信号为高电平时进行解码
          tot <= tot + 1'b1; //一共解了多少个码
          case(tot) //根据当前的解码个数将输出信号赋给totout信号的对应位数
            //0，1，2，3，4，5，6，7，8
            8:light3 <= 1'b1; //已经超出8位解码，长度超限报错
          endcase
          code <= 5'b0;
          th <= 3'b0;
          length <= 3'b0;
```

```
        light2 <= 1'b0;
    end
    else if(fshort == 1'b1)begin // 当短键按下时更新长短码缓冲区code值
         case(length) // 根据当前长短码输入的长度对输出信号进行更新
            3'b000: out<=8'b0000_0110;//e
            3'b001:
                case(code[0])1'b0//i,  1'b1//n
            3'b010:
                case(code[1:0]) 2'b10//r,  2'b00//s,  2'b11//g,  2'b01//d
            3'b011:
                case(code[2:0])
                  //分别给out赋值b,z,c,l,h,f,p的摩斯电码数码管内容
                  //省略多个具体case......
                  default: begin //无法解码时为空白输出
                    out<=8'b1111_1111;
                    light2 <= 1'b1;//无法解码时报错
                  end
                endcase
             end
            3'b100
                case(code[3:0])
                  //分别给out赋值8,6,5,7,9的摩斯电码数码管内容
                  //省略多个具体case......
                  default:
                    out<=8'b1111_1111;//无法解码时为空白输出
                    light2 <= 1'b1;//无法解码时报错
                endcase
           default : out <= 8'b11111111;//无
        endcase
        th <= th + 1'b1;
        length <= length + 1'b1;
    end
      else if(flong == 1'b1)begin // 当长键按下时更新长短码缓冲区code值
            //对length进行分case说明，形式同short
            //分别给out赋值a,m,w,u,o,k,x,q,y,v,j1,1,3,4,0的摩斯电码数码管内容
            //省略多个具体case......
           case(length) // 根据当前长短码输入的长度对输出信号进行更新
             //具体case：3'b000，3'b001，3'b010，3'b011，3'b100
             default : out <= 8'b11111111;//无
           endcase
           code[th] <= 1'b1;
           th <= th + 1'b1;
           length <= length + 1'b1;
      end
      else if (fclear == 1'b1)//当clear按键按下时重置所有变量
           length <= 3'b0; code <= 5'b0;  tot <= 4'b0;  th <= 3'b0;
           totout <= 64'b0;  out <= 8'b0;  light1 <= 1'b0;
           light2 <= 1'b0;  light3 <= 1'b0;

      else if(fback == 1'b1)begin //当back按键按下时清除长短码缓冲区最后一位
           if(th != 0)begin
                 light1 <= 1'b0;
              case(length)
                 3'b001:begin
```

```
                    out <= 8'b11111111;
                end
                3'b010:
                    case(code[0]) 1'b0//e   1'b1//t endcase
                3'b011:begin
                    case(code[1:0]) 2'b00//i  2'b01//n  2'b10//a  2'b11//m
                end
                3'b100:begin
                    case(code[2:0])//省略部分代码，只显示大概框架
                        3'b000//s ,3'b001//d ,3'b010//r ,3'b011//g
                        3'b100//u, 3'b101//k, 3'b110//w, 3'b111//o
                        default
                            out <= 8'b11111111;//无法解码时为空白输出
                            light2 <= 1'b1;//无法解码时报错
                    endcase
                end
                3'b110:begin
                  case(code[3:0])4'b0000//h,  4'b0001//b,  4'b0010//l,
                    4'b0011//z,  4'b0100//f,  4'b0101//c, 4'b0110//p,
                    4'b1000//v,,  4'b1001//x, 4'b1011//q，4'b1101//y,
                    4'b1110//j
                    default：
                      out<= 8'b1111_1111;//无法解码时为空白输出
                      light2 <= 1'b1;//无法解码时报错
                  endcase
                end
                endcase
                code[th - 1'b1] <= 1'b0;
                th <= th - 1'b1;
                length <= length - 1'b1;
            end
            else  light1 <= 1'b1; //长度为0无法继续退格报错
          end
    end
end
```

▼ 扫描显示模块

```
always @( scan_cnt)
      begin
      case ( scan_cnt )
          4'b0001 : seg_en = 8'b1111_1110;
          4'b0010 : seg_en = 8'b1111_1101;
          4'b0011 : seg_en = 8'b1111_1011;
          4'b0100 : seg_en = 8'b1111_0111;
          4'b0101 : seg_en = 8'b1110_1111;
          4'b0110 : seg_en = 8'b1101_1111;
          4'b0111 : seg_en = 8'b1011_1111;
          4'b1000 : seg_en = 8'b0111_1111;
          default : seg_en = 8'b1111_1111;
```

```
       endcase
       end

always @ (scan_cnt )
     begin
     case (scan_cnt)
         1: Y_r = totout[7:0];
         2: Y_r = totout[15:8];
         3: Y_r = totout[23:16];
         4: Y_r = totout[31:24];
         5: Y_r = totout[39:32];
         6: Y_r = totout[47:40];
         7: Y_r = totout[55:48];
         8: Y_r = totout[63:56];
         default: Y_r = 8'b11111111;
     endcase
     end
```

板上测试

见视频

总结及优化

- 问题及解决方案

  ▼ 矩阵键盘消抖

  利用分频信号+计数器延迟10ms进行消抖

  利用key_filter_cnt计数器对每次按键进行10ms延迟操作、同时对得到周期以1000000ns一周期的clk1k周期信号

```
//key_filter_cnt,按键防抖计数器
reg [3:0] key_filter_cnt;
always @ (posedge clk1k or posedge rst)
  if (rst)
    key_filter_cnt <= 0;
  else
    case (next_state)
      NO_KEY_PRESSED :// key_filter_cnt=0
    key_filter_cnt <= 0;
      SCAN_COL0 : // 如果按键是col0的，则防抖时间+1，否则回0
        if(row != 4'hF)
```

```
      key_filter_cnt <= key_filter_cnt+1;
    else
      key_filter_cnt <= 0;
      SCAN_COL1 :// 如果按键是col1的，则防抖时间+1，否则回0
        if(row != 4'hF)
      key_filter_cnt <= key_filter_cnt+1;
    else
      key_filter_cnt <= 0;
      SCAN_COL2 : // 如果按键是col2的，则防抖时间+1，否则回0
        if(row != 4'hF)
      key_filter_cnt <= key_filter_cnt+1;
    else
      key_filter_cnt <= 0;
      SCAN_COL3 :// 如果按键是col3的，则防抖时间+1，否则回0
        if(row != 4'hF)
      key_filter_cnt <= key_filter_cnt+1;
    else
      key_filter_cnt <= 0;
      KEY_PRESSED :// key_filter_cnt=0
    key_filter_cnt <= 0;
    endcase
```

- 列扫描+key_filter_cnt延迟消抖

```
//key_filter_cnt计数到9，信号生效
always @ (*)
  case (current_state)
    NO_KEY_PRESSED :// 初始态，所有列选置0，等待按键按下后，先进入col0
        if (row != 4'hF)
          next_state = SCAN_COL0;
        else
          next_state = NO_KEY_PRESSED;
    SCAN_COL0 :
    // 如果按键在col0，则进行防抖10ms，成功进入KEY_PRESSED，否则进入col1
        if (row != 4'hF)
      if (key_filter_cnt==9)
        next_state = KEY_PRESSED;
      else
        next_state = SCAN_COL0;
        else
          next_state = SCAN_COL1;
    SCAN_COL1 :
    // 如果按键在col1，则进行防抖10ms，成功进入KEY_PRESSED，否则进入col2
        if (row != 4'hF)
      if (key_filter_cnt==9)
        next_state = KEY_PRESSED;
      else
        next_state = SCAN_COL1;
        else
          next_state = SCAN_COL2;
    SCAN_COL2 :
```

```
    // 如果按键在col2，则进行防抖10ms，成功进入KEY_PRESSED，否则进入col3
        if (row != 4'hF)
      if (key_filter_cnt==9)
        next_state = KEY_PRESSED;
      else
        next_state = SCAN_COL2;
        else
          next_state = SCAN_COL3;
    SCAN_COL3 :
// 如果按键在col3，则进行防抖10ms，成功进入KEY_PRESSED，否则进入NO_KEY_PRESSED
        if (row != 4'hF)
      if (key_filter_cnt==9)
        next_state = KEY_PRESSED;
      else
        next_state = SCAN_COL3;
        else
          next_state = NO_KEY_PRESSED;
    KEY_PRESSED : // 按键全为f后，代表按键松开，进入NO_KEY_PRESSED
        if (row != 4'hF)
          next_state = KEY_PRESSED;
        else
          next_state = NO_KEY_PRESSED;
  default : next_state = NO_KEY_PRESSED;
  endcase
```

▼ 解码模块按键消抖

设计一个key_filter模块对按键反应，产生一个分频周期的脉冲信号，计数达到20ms，进行消抖

```
//***************************** Main Code ****************************//
//cnt_20ms:如果时钟的上升沿检测到外部按键输入的值为低电平时，计数器开始计数
always@(posedge sys_clk or posedge sys_rst_n)begin
    if(sys_rst_n == 1'b1)
        cnt_20ms <= 20'b0;
    else    if(key_in == 1'b1)
        cnt_20ms <= 20'b0;
    else    if(cnt_20ms == CNT_MAX && key_in == 1'b0)
        cnt_20ms <= cnt_20ms;
    else
        cnt_20ms <= cnt_20ms + 1'b1;
end
//key_flag:当计数满20ms后产生按键有效标志位
//且key_flag在999_999时拉高,维持一个时钟的高电平
always@(posedge sys_clk or posedge sys_rst_n)begin
    if(sys_rst_n == 1'b1)
        key_flag <= 1'b0;
    else    if(cnt_20ms == CNT_MAX - 1'b1)
        key_flag <= 1'b1;
    else
```

```
        key_flag <= 1'b0;
end
```

▼ 合并模块时出现multi-driver问题

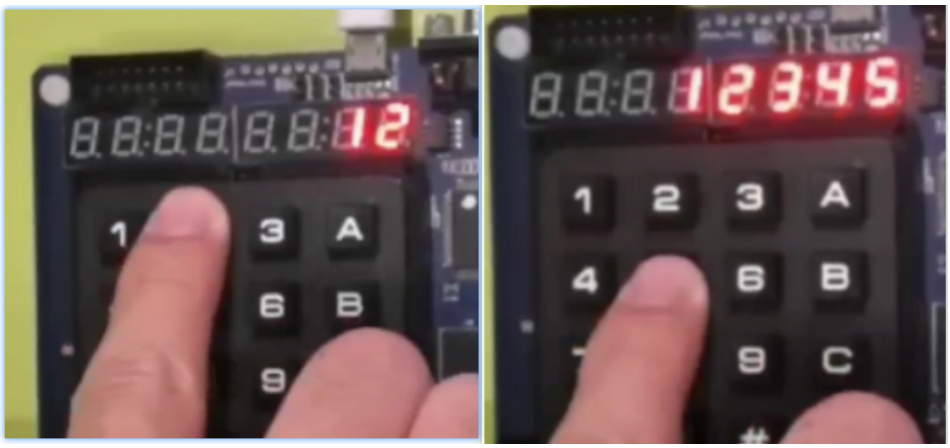
将两个模块合并在同一个文件中，将所涉及的信号合并在一个同一个always语句块

▼ 解码模块非阻塞赋值同时进行与其他操作的冲突

改变判断条件，以顺利执行

- 系统特色

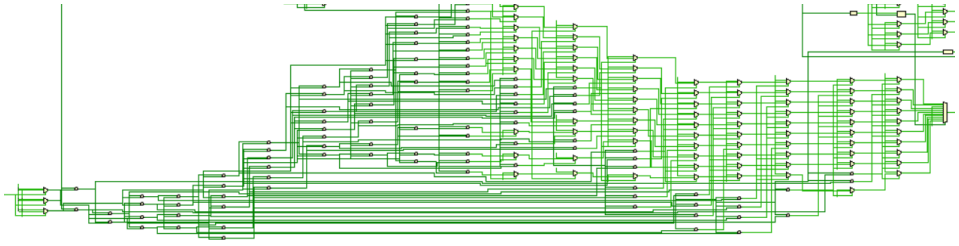
▼ 矩阵键盘输入显示为逐个移位，滑动显示



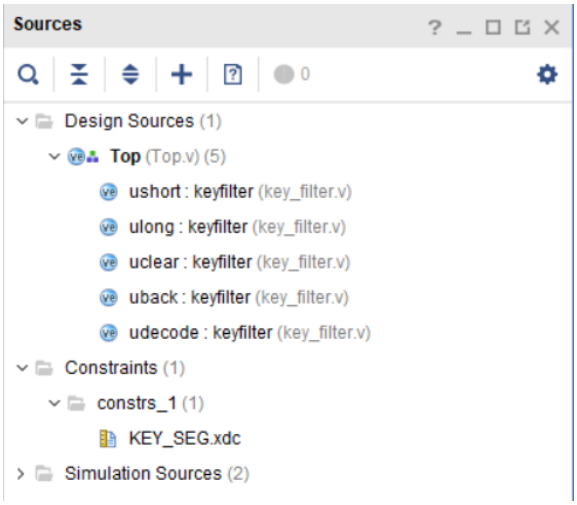
▼ 实现了编码、解码模块各类操作，如退格、删除、清空、错误报警等功能

- 优化方向

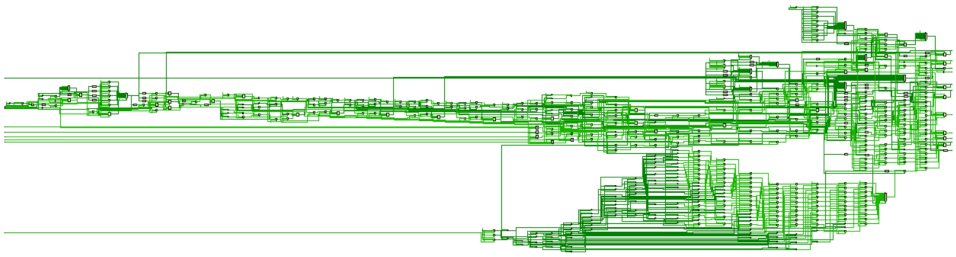
▼ 蜂鸣器部分使用了较多的if-else语句，可以简化写法，使电路情况较为冗余复杂



▼ 所有代码合并在一个文件中，代码模块结构不够清晰直观



RTL分析 （Top.v设计文件）



▼ 端口绑定比较随意，可以再整理一下