

SUSTech_CS207_Final-Project_2021F

小组成员: 安钧文, 张嘉浩, 廖子良

开发计划

1. 小组选题: 摩斯电码

2. 成员分工及贡献百分比

- 安钧文: 解码模块全部逻辑, 切换状态指示灯, 项目总合成debug [@wanteatfruit](#)
- 张嘉浩: 顶层模块设计, 编码与解码中数码管和小键盘的实现, 流程图&项目报告合成 [@MichaelZhangjiahao](#)
- 廖子良: 编码模块蜂鸣器全部, 编码模块合成, 编码模式可一键放歌

(各成员所负责部分均完成全部bonus)

3. 进度安排

1. 项目分析&基础功能实现 11.14~11.28
2. 完成bonus 11.28~12.5
3. 项目合成 12.5~12.12
4. 报告及整体完善 12.12~12.19

4. 执行记录

- 11/14 开始讨论project, 确定选题为停车场收费系统
- 11/28 画完停车场收费系统project的流程图, 发现难度较大, 开始考虑其他project
- 11/29 画完摩斯电码project流程图, 召开“遵义会议”, 决定转做摩斯电码
- 12/4 开始研究小键盘和数码管的工作原理
- 12/4 开始对解码模块构思, 学习分频器、按键消抖、状态机
- 12/8 开始处理小键盘代码
- 12/10 正式着手解码模块的编程
- 12/12 开始处理七段数码管代码
- 12/13 找到蜂鸣器播放的原理并且着手开始写蜂鸣器模块
- 12/14 完成输入长短码、退格的功能, 并能正确显示在LED中
- 12/15 写了两种实现方式, 其中一种能够播放目标声音
- 12/16 写完小键盘和数码管分别代码
- 12/20 实现小键盘输入进数码管, 以及一键清空数码管
- 12/20 完成解码逻辑, 能正确将LED中的摩斯电码解码并显示于数码管中
- 12/22 在与队友代码合成之后, 能够在板子上通过八个拨码开关播放出来数字对应的声音, 新增长模式(长短码以及间隔声音变长一倍)。如果该位置上没有需要编码的字符, 则蜂鸣器不输出。
- 12/24 实现小键盘的退格
- 12/25 对小键盘的退格键增加了防抖模块
- 12/25 蜂鸣器增加了一键播放功能
- 12/25 完成对不能解码的情况、七段数码管已满的情况的错误提示
- 12/27 对项目进行初步合成, 合成失败
- 12/28 成功加入了不同模式的状态指示灯
- 12/28 项目合成成功!
- 12/29 尝试新增播放音乐功能, 但引发了其他地方的问题

- 12/30 项目再次合成成功！

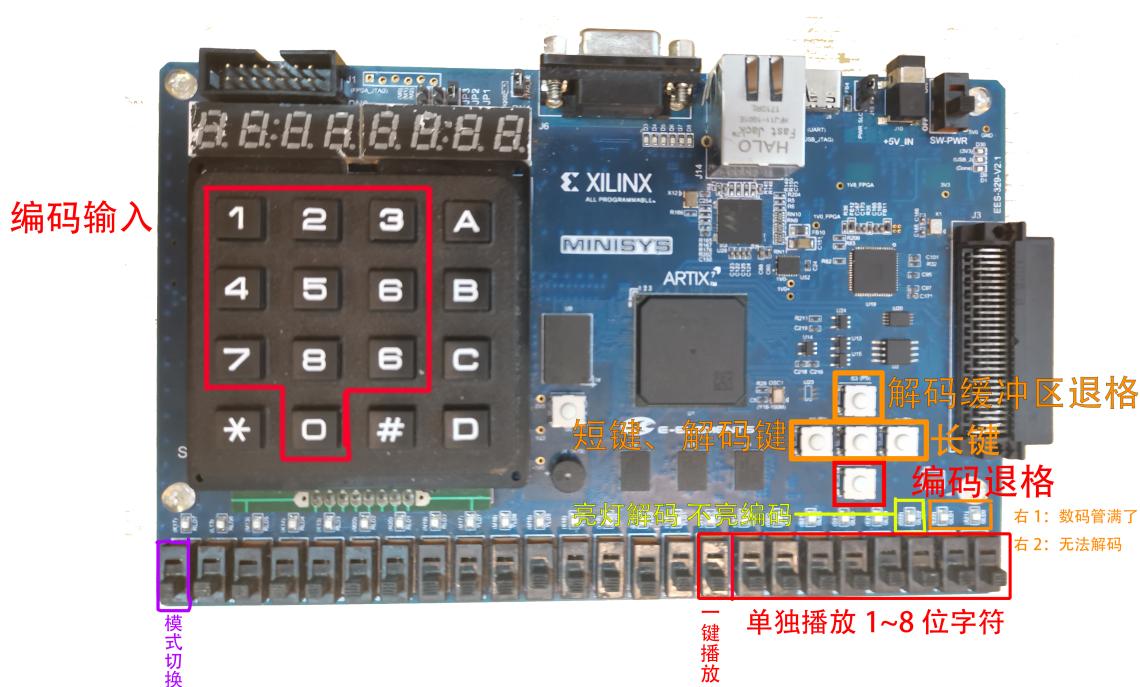
总结：

在完成project的过程中，由于开始对Verilog的不熟练，以及出现了更换项目选题等意外状况，在project初期的拖延现象较为严重。但得益于lab课上老师对分频器和状态机的细致讲解，逐渐对时序逻辑有了更深入的理解之后，项目进度稳步推进，较为圆满地完成了这次的project任务。

5. 项目视频展示

[摩斯电码project演示](#)

6. 用户操作图解



项目设计

1. 需求分析

- 系统功能：

本项目实现了一个摩斯电码收发器（下称“收发器”），该收发器具有两种模式，能分别支持摩斯电码的编码和解码。

收发器在解码模式中，能接受长短码的输入、能够在24个LED中显示输入的电码、以及能够在七段数码管中显示解码后的电码。

收发器在编码模式中，能接受小键盘输入，能够将输入的数字显示在七段数码管中（最多8字符）、能够将编码后的电码通过蜂鸣器按照一定规律播放出来。

此外，在解码模式中，长短码缓冲区拥有退格功能；而当解码出现错误时，均有不同的错误警告以告诉用户错误类型。在编码模式中，七段数码管最后输入的字符可以实现退格，且蜂鸣器播放速度可分长短模式播放。除了单个播放字符以外，此模块还具备一键播放全部字符对应摩斯电码的功能，还可以在解码时可以一键播放动听的歌曲。（本项目所有的按键均已实现防抖功能，极大地方便了用户的使用）

- 顶层端口规格：

输入端：系统时钟信号clk；一键播放开关key_on_all；播放音乐开关key_on_mus；控制蜂鸣器播放摩斯电码key_on1~key_on8；长短模式切换modulee；小键盘行值row[3:0]；复位信号rst；小键盘退格键up；解码长短码缓冲区退格键down；解码编码模式切换开关switch；长码键long；解码键mid；短码键short

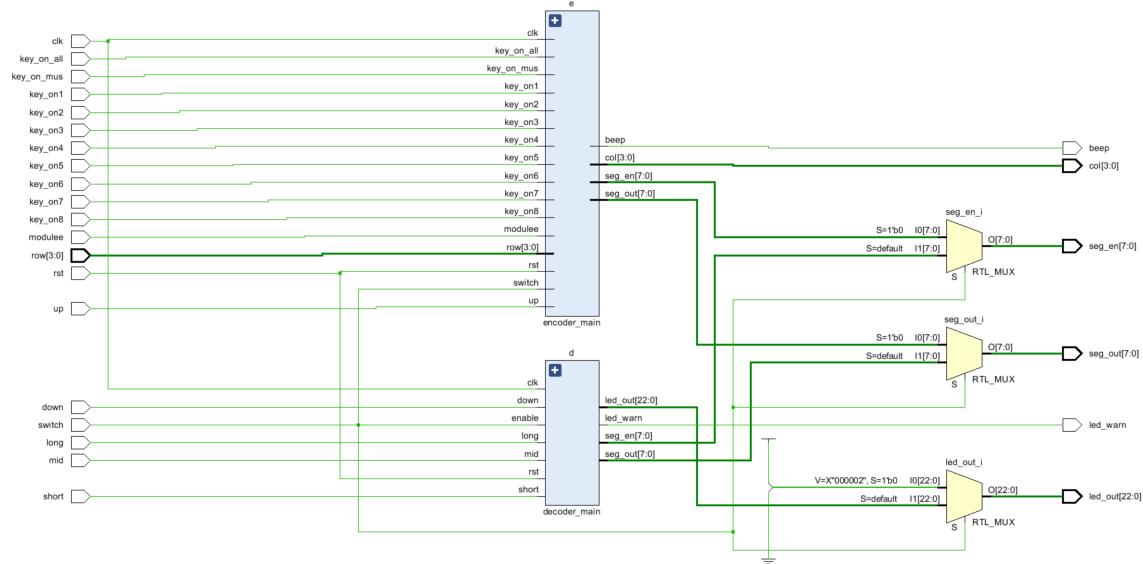
输出端：蜂鸣器输出beep；小键盘扫描得到列值输出col；七段数码管使能信号seg_en[7:0]；七段数码管输出方式控制seg_out[7:0]；长短码缓冲区的LED提示led_warn；LED输出接口led_out[22:0]

- **开发板输入设备：**数字小键盘，拨码开关
- **开发板输出设备：**七段数码管，led灯，蜂鸣器

2. 系统结构设计

各模块接口：

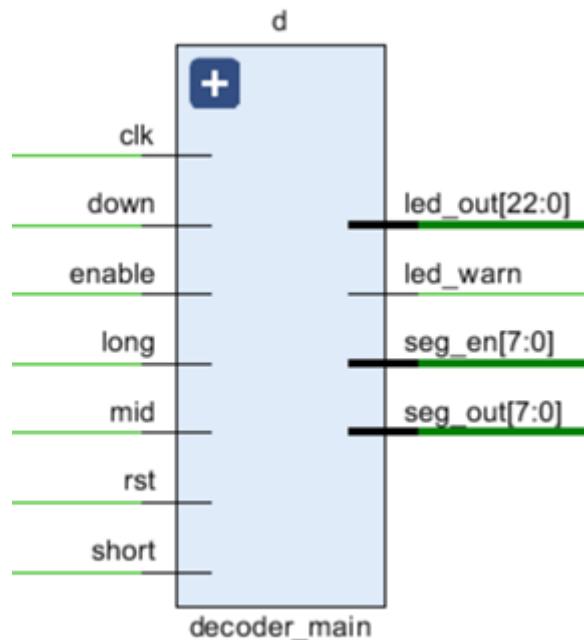
顶层模块逐个接口的讲解见上方“顶层端口”部分，且在encoder_main与decoder_main的模块中有着更为详细的说明讲解



• 解码

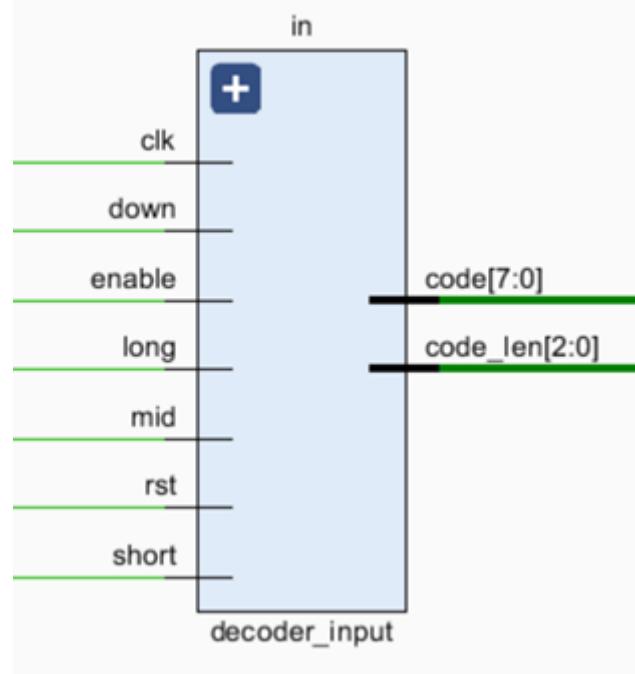
输入接口clk是时钟信号，down, long, mid, rst, short分别对应退格键、长键、解码键、重置键、短键。Enable为使能信号。

输出接口led_out[22:0]、led_warn是长短码缓冲区对应的LED显示（包含错误警告），seg_en控制八位数码管是否显示，seg_out控制每个数码管的显示内容。



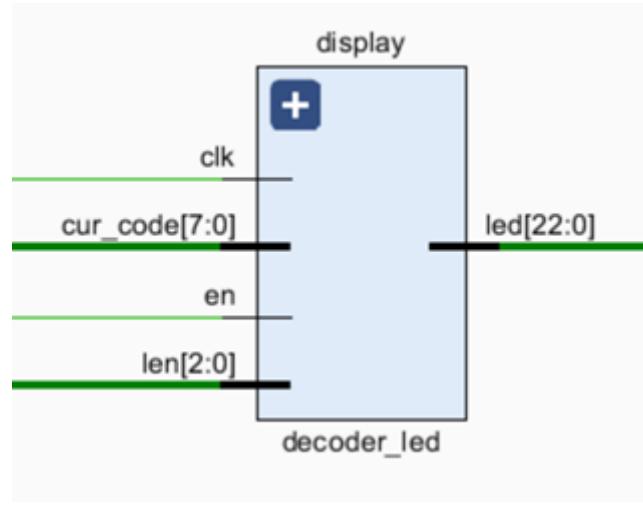
输入接口clk是时钟信号，down, long, mid, rst, short分别对应退格键、长键、解码键、重置键、短键。
Enable为使能信号。（与decoder_main模块相同）

输出接口code[7:0]为单个摩斯电码输入（code[5:7]为它的长度，code[4:0]为它的长短码类别），
code_len为单个摩斯电码的长度，与code[5:7]内容相同。



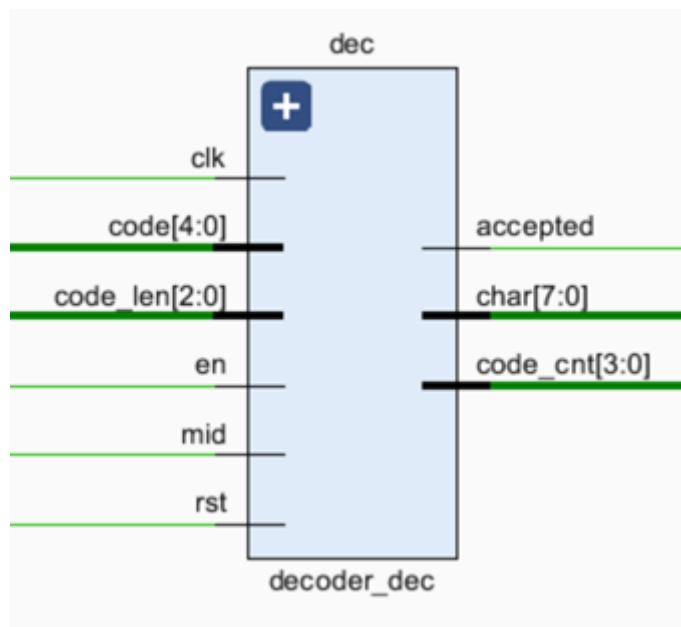
输入：时钟和对应的单个摩斯电码的总输入、它的长度、使能信号

输出：对应的LED显示（只包含无法解码的警告）



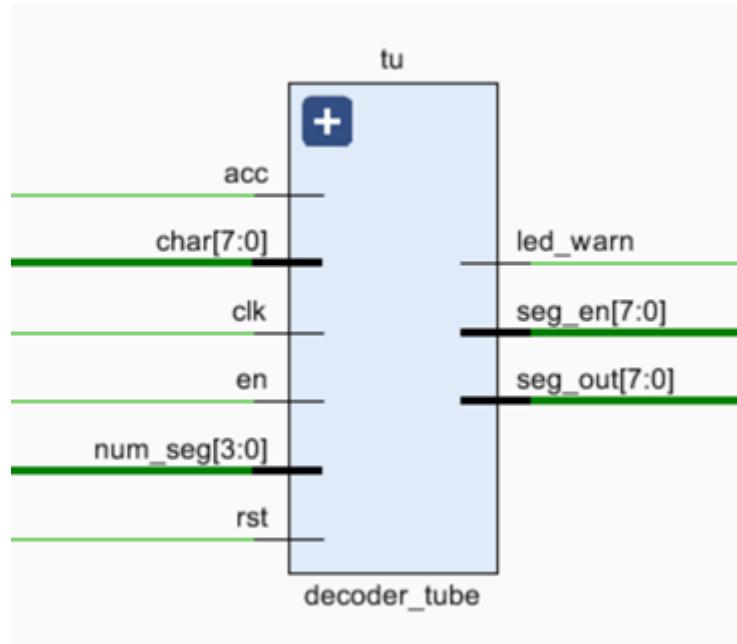
输入端口：时钟信号、code[4:0]为长短码类别、code_len[2:0]为长短码长度（code为input模块输出的低5位，code_len为高3位）、使能信号、解码键、复位键

输出端口：accepted为解码是否合法的信号、char[7:0]解码单个电码后需要在数码管显示的内容、code_cnt[3:0]为已经解码了字符的个数



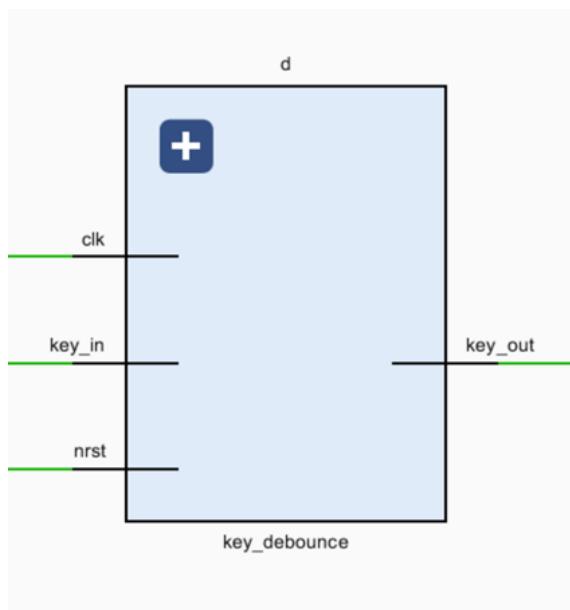
输入端口：Acc为输入的解码是否合法，char[7:0]为当前需要在数码管中显示的内容，clk为时钟，en为使能信号，num_seg[3:0]为当前在数码管中已经显示的数字个数（已经解码了多少电码），rst为数码管复位信号。

输出端口：led_warn为数码管已满时对应到led灯的警告，seg_en控制八位数码管是否显示，seg_out控制每个数码管的显示内容。



输入端口：时钟信号、需要消抖的按键、复位信号

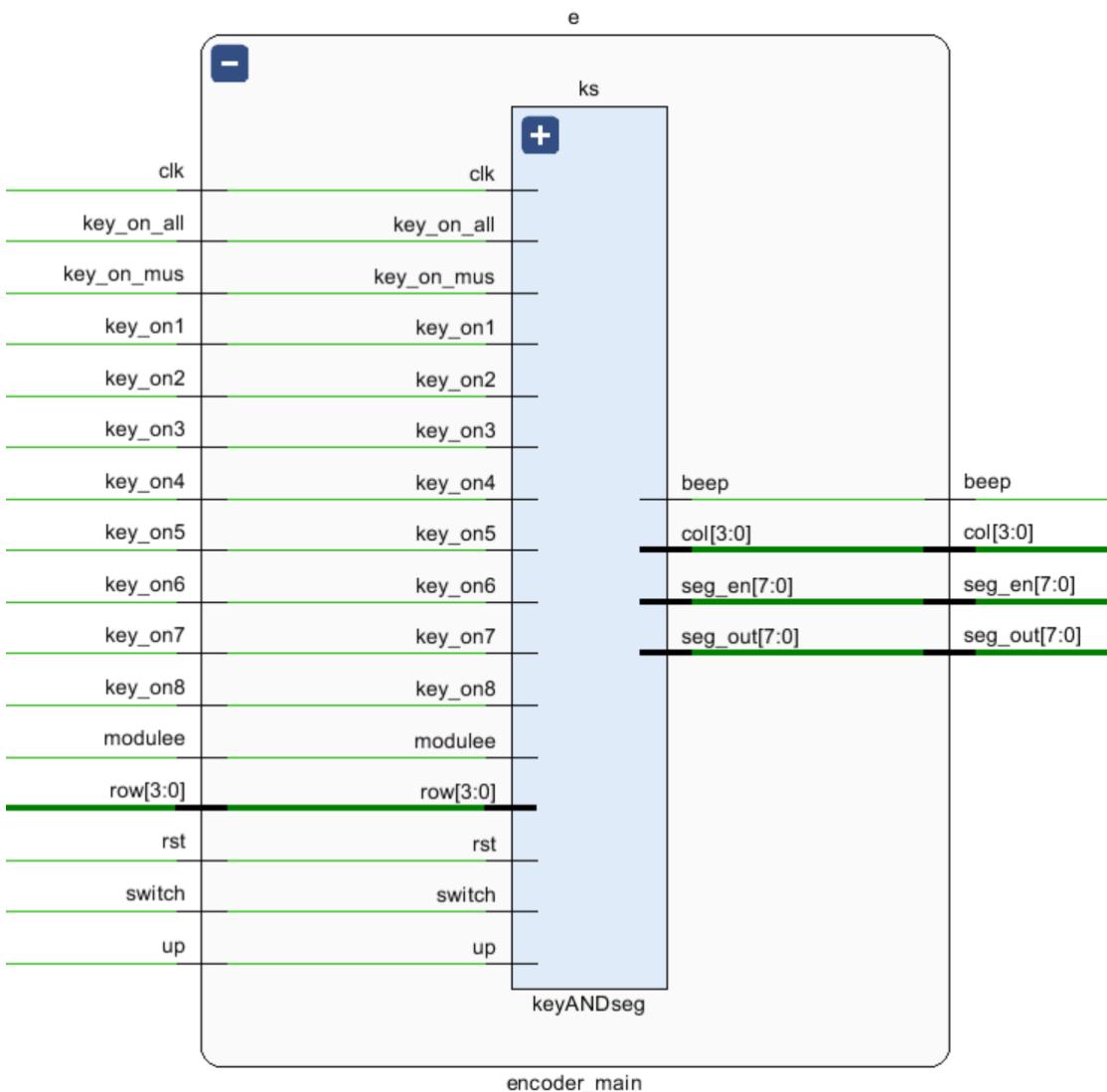
输出端口：消抖后的信号



• 编码

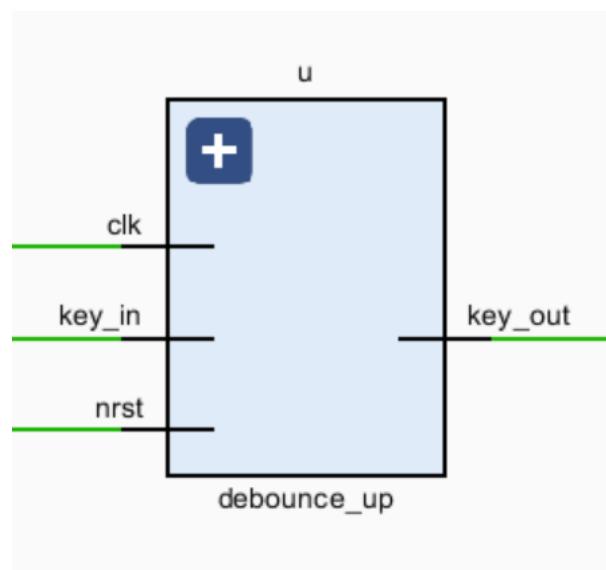
输入端口：clk是时钟信号，key_on_all判断是否一键播放所有电码，key_on_mus判断是否播放音乐，key_on1到key_on8分别判断从左到右的八个七段数码管是否播放对应的电码，modulee为长短模式切换，row是扫描小键盘时的行值，rst是复位信号（用于清空数码管），switch是切换解码编码模式的开关，up是七段数码管退格键的绑定

输出端口：beep是蜂鸣器输出信号，col是扫描小键盘得到的列值，seg_en是控制七段数码管在某一时刻哪几盏灯会亮，seg_out是控制每盏灯的亮法以显示不同的数字



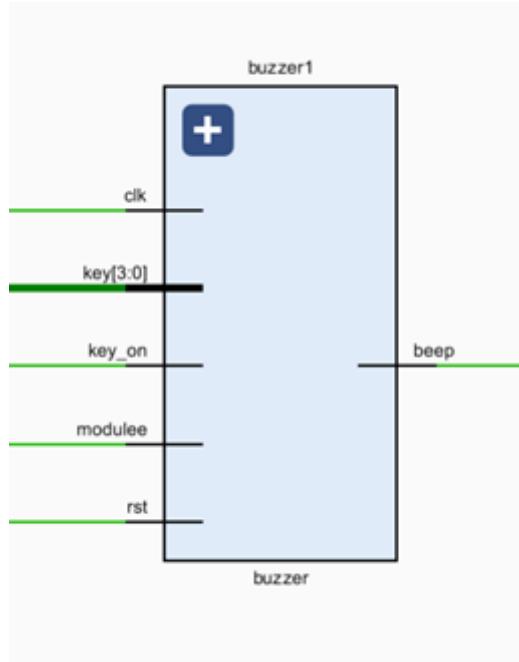
输入端口：clk为时钟信号，需要消抖的up按键，复位信号

输出端口：消抖后的信号

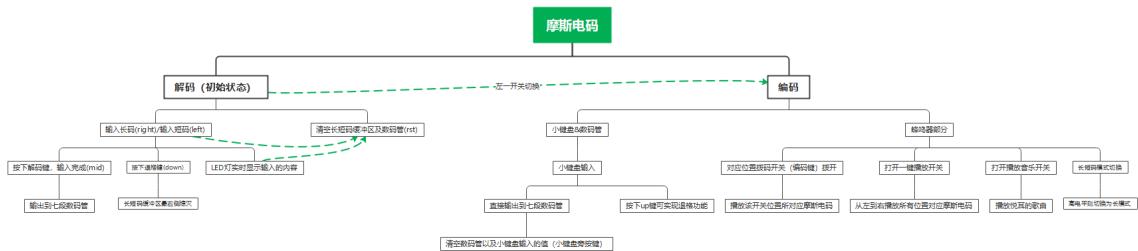


输入端口：key为该七段数码管位置的值，key_on用于判断是否播放该位置的值所代表的电码，modulee为长短模式切换，clk为时钟信号，rst为重置键

输出端口：beep是蜂鸣器输出信号



状态迁移流程图：



3. 详细设计

模块名	核心功能	备注
top	汇总解码编码的顶层模块	对七段数码管用MUX以避免multi-driver的错误
decoder_main	解码模块顶层模块	
decoder_input	解码模块的按键输入模块	长键、短键、退格键、解码键、重置键
decoder_led	解码模块的LED显示模块	实时显示当前缓冲区内容
decoder_dec	解码模块的“解码”模块	按下解码键后将缓冲区清空，输出对应字符
decoder_tube	解码模块的数码管显示模块	将解码的字符显示到数码管中
key_debounce	按键消抖模块	
encoder_main	编码顶层模块	对keyANDseg进行绑定以避免逻辑紊乱
keyANDseg	汇总小键盘和数码管模块	实现小键盘在数码管中的输入
debounce_up	退格“up”键消抖模块	
buzzer	蜂鸣器模块	根据数码管显示内容以及播放开关开闭，播放对应音频

核心代码及说明：

顶层模块

```
//绑定编码的顶层模块encoder_main以及解码的顶层模块decoder_main

module top (
    input switch,
    input clk,
    input rst,
    input down,
    input short,
    input long,
    input mid,
    input up,      //退格
    input [3:0] row,           // 矩阵键盘 行
    input
    key_on1, key_on2, key_on3, key_on4, key_on5, key_on6, key_on7, key_on8, key_on_a11, key_on_a12,
    input moduleee,
    output [3:0] col,          // 矩阵键盘 列
    output reg [22:0] led_out,
    output led_warn,
    output reg [7: 0] seg_en,
    output reg [7: 0] seg_out,
    output beep);

```

```

wire [7:0] seg_en_ecd;
wire [7:0] seg_out_ecd;
wire [7:0] seg_en_dcd;
wire [7:0] seg_out_dcd;
wire[22:0] led_dec;
reg [22:0] led_enc=23'b00000_00000_00000_00000_010;

encoder_main e(clk, rst, up, row,
key_on1,key_on2,key_on3,key_on4,key_on5,key_on6,key_on7,key_on8,
key_on_all,key_on_mus,modulee,switch,col,
seg_en_ecd,seg_out_ecd,beep);
decoder_main d(clk, switch, down, short, long, mid, rst, led_dec,//one light
for short, 2 for long, separate 1 between
led_warn,seg_en_dcd, seg_out_dcd);

always @(switch) begin
  if (!switch) begin //encoder
    seg_en = seg_en_ecd;
    seg_out = seg_out_ecd;
    led_out=led_enc;
  end
  else begin //decoder
    seg_en = seg_en_dcd;
    seg_out = seg_out_dcd;
    led_out=led_dec;
  end
end
endmodule

```

解码部分

解码分频器

```

//解码模式input, led, dec模块的分频器

reg [23:0] cnt1;
reg clk_out;
always @(posedge clk)
begin
  if(!enable)begin
    cnt1 <= 0;
    clk_out <= 0;
  end
  else begin
    if(cnt1 == 9500000)begin
      clk_out <= ~clk_out;
      cnt1 <= 0;
    end
    else
      cnt1 <= cnt1+1;
  end
end

```

解码输入模块 decoder_input

```
//已输入电码长度(cur_cnt)和总电码(cur_code, 高3位记录长度, 低5位记录类型)的状态机
```

```
always @ (posedge clk_out, negedge enable) begin
    if (!enable) begin //init
        cur_cnt <= 3'b000;
        cur_code <= 8'b0000_0000;
    end
    else
        cur_cnt <= tot_cnt;
        cur_code <= total_code;
end
```

```
//改变状态的组合逻辑部分, 根据当前电码长度, 判断下一、退格后的电码长度, 和需要的记录的长短码类型
```

```
always@* begin
    case (cur_cnt)
        cnt_0: if (s_out) begin tot_cnt = cnt_1; end
            else if (!out) begin tot_cnt = cnt_1; total_code[0] = 1; end
            else if (rst | m_out) begin tot_cnt = 3'b000; total_code = 8'b0000_0000;
    end
        else if (d_out) tot_cnt = cnt_0;
        else begin tot_cnt = cnt_0; {total_code[4:0]} = {cur_code[4:0]}; end
    //change
        cnt_1: if (s_out) begin tot_cnt = cnt_2; end
            else if (!out) begin tot_cnt = cnt_2; total_code[1] = 1; end
            else if (rst | m_out) begin tot_cnt = 3'b000; total_code = 8'b0000_0000;
    end
        else if (d_out) begin tot_cnt = cnt_0; total_code[0] = 0; end
        else begin tot_cnt = cnt_1; {total_code[4:0]} = {cur_code[4:0]}; end
        cnt_2: if (s_out) begin tot_cnt = cnt_3; end
            else if (!out) begin tot_cnt = cnt_3; total_code[cnt_2] = 1; end
            else if (rst | m_out) begin
                tot_cnt = 3'b000; total_code = 8'b0000_0000; end
            else if (d_out) begin tot_cnt = cnt_1; total_code[cnt_1] = 0; end
            else begin tot_cnt = cnt_2; {total_code[4:0]} = {cur_code[4:0]}; end
        cnt_3: if (s_out) begin tot_cnt = cnt_4; end
            else if (!out) begin tot_cnt = cnt_4; total_code[cnt_3] = 1; end
            else if (rst | m_out) begin tot_cnt = cnt_0; total_code = 8'b0000_0000;
    end
        else if (d_out) begin tot_cnt = cnt_2; total_code[cnt_2] = 0; end
        else begin tot_cnt = cnt_3; {total_code[4:0]} = {cur_code[4:0]}; end
        cnt_4: if (s_out) begin tot_cnt = cnt_5; end
            else if (!out) begin tot_cnt = cnt_5; total_code[cnt_4] = 1; end
    //total[4] is the fifth bit.
            else if (rst | m_out) begin tot_cnt = cnt_0; total_code = 8'b0000_0000; end
            else if (d_out) begin tot_cnt = cnt_3; total_code[cnt_3] = 0; end
            else begin tot_cnt = cnt_4; {total_code[4:0]} = {cur_code[4:0]}; end
        cnt_5: if (s_out | !out) begin tot_cnt = cnt_0; total_code = 8'b0000_0000;
    end
        else if (rst | m_out) begin tot_cnt = cnt_0; total_code = 8'b0000_0000; end
        else if (d_out) begin tot_cnt = cnt_4; total_code[cnt_4] = 0; end
        else begin tot_cnt = cnt_5; {total_code[4:0]} = {cur_code[4:0]}; end
    //but counted 5 bits, aware the difference
    endcase
```

```
end
```

```
//根据当前电码长度和按键输入，改变总电码的高3位，即记录电码长度的3位
```

```
always @(*) begin
    case(cur_cnt)
        3'b000: if(s_out | l_out)
            {total_code[7],total_code[6],total_code[5]}=3'b001;
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else {total_code[7],total_code[6],total_code[5]}=3'b000;
        3'b001: if(s_out | l_out) begin
            {total_code[7],total_code[6],total_code[5]} =3'b010; end
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(d_out)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else {total_code[7],total_code[6],total_code[5]} =3'b001;
        3'b010: if(s_out | l_out) begin
            {total_code[7],total_code[6],total_code[5]} =3'b011; end
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(d_out)
            {total_code[7],total_code[6],total_code[5]}=3'b001;
        else {total_code[7],total_code[6],total_code[5]} =3'b010;
        3'b011: if(s_out | l_out)
            {total_code[7],total_code[6],total_code[5]}=3'b100;
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(d_out)
            {total_code[7],total_code[6],total_code[5]}=3'b010;
        else {total_code[7],total_code[6],total_code[5]}=3'b011;
        3'b100: if(s_out | l_out)
            {total_code[7],total_code[6],total_code[5]}=3'b101;
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(d_out)
            {total_code[7],total_code[6],total_code[5]}=3'b011;
        else {total_code[7],total_code[6],total_code[5]} =3'b100;
        3'b101: if(s_out | l_out)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(m_out|rst)
            {total_code[7],total_code[6],total_code[5]}=3'b000;
        else if(d_out)
            {total_code[7],total_code[6],total_code[5]}=3'b100;
        else {total_code[7],total_code[6],total_code[5]}=3'b101;
    endcase
end
```

解码LED显示模块 decoder_led

```
//改变LED输出的状态机

always@(posedge clk_out, negedge en)
begin
    if(~en)
        begin
            cur_state<= 23'b0000_0000_0000_0000_0000;
        end
    else
        cur_state <= nxt_state;
end
```

//根据电码长度(len)、电码类型(cur_code[len-1:0])，改变LED状态
//枚举了每种可能的输入，注释中l 表示长码， s 表示短码， N表示无法解码， E表示可以解码

```
always @*
begin
    case(len)
        default: nxt_state=23'b00000_00000_00000_00000_000;
        len_1: case (cur_code[0])
            1'b1: nxt_state=23'b11000_00000_00000_00000_000; //1 long
            1'b0: nxt_state=23'b10000_00000_00000_00000_000; //1 short
        endcase
        len_2: case ({cur_code[1],cur_code[0]})
            2'b00:nxt_state=23'b10100_00000_00000_00000_000; //short short
            2'b01: nxt_state=23'b11010_00000_00000_00000_000; //long short
            2'b10:nxt_state=23'b10110_00000_00000_00000_000; //short long
            2'b11:nxt_state=23'b11011_00000_00000_00000_000; //long long
        endcase
        len_3: case ({cur_code[2],cur_code[1],cur_code[0]})
            3'b111:nxt_state=23'b11011_01100_00000_00000_000;
            3'b000:nxt_state=23'b10101_00000_00000_00000_000;
            3'b001:nxt_state=23'b11010_10000_00000_00000_000;//l s s
            3'b010:nxt_state=23'b10110_10000_00000_00000_000;//s l s
            3'b011:nxt_state=23'b11011_01000_00000_00000_000;//l l s
            3'b100:nxt_state=23'b10101_10000_00000_00000_000;//s s l
            3'b110:nxt_state=23'b10110_11000_00000_00000_000;//s l l
            3'b101:nxt_state=23'b11010_11000_00000_00000_000;//l s l
        endcase
        len_4: case ({cur_code[3],cur_code[2],cur_code[1],cur_code[0] })
            4'b1111:nxt_state=23'b11011_01101_10000_00000_001;//N.
            4'b0000:nxt_state=23'b10101_01000_00000_00000_000;// E.
            4'b0001:nxt_state=23'b11010_10100_00000_00000_000;//lsss E
            4'b0010:nxt_state=23'b10110_10100_00000_00000_000;//s1ss E
            4'b0100:nxt_state=23'b10101_10100_00000_00000_000;//ss1s E
            4'b1000:nxt_state=23'b10101_01100_00000_00000_000;//sss1 E
            4'b0011:nxt_state=23'b11011_01010_00000_00000_000;//l1ss E
            4'b0110:nxt_state=23'b10110_11010_00000_00000_000;//s1ls E
            4'b1100:nxt_state=23'b10101_10110_00000_00000_001;//ss1l N
            4'b0101:nxt_state=23'b11010_11010_00000_00000_000;//lsls E
            4'b1010:nxt_state=23'b10110_10110_00000_00000_001;//s1s1 N
            4'b1001:nxt_state=23'b11010_10110_00000_00000_000;//lss1 E
            4'b0111:nxt_state=23'b11011_01101_00000_00000_001;//llls N
            4'b1101:nxt_state=23'b11010_11011_00000_00000_000;//ls11 E
            4'b1011:nxt_state=23'b11011_01011_00000_00000_000;//lls1 E
            4'b1110:nxt_state=23'b10110_11011_00000_00000_000;//s111 E
        endcase
    endcase
end
```

```

        default: nxt_state=23'b11111_11111_11111_11111_111; //wrong code
warning
    endcase
    len_5:
case({cur_code[4], cur_code[3], cur_code[2], cur_code[1], cur_code[0] })
    //if the user inputs wrongly 5bit inputs, can set up a error
display. (4 bits also)
    5'b1111:nxt_state=23'b11011_01101_10110_00000_000;
    5'b11110:nxt_state=23'b10110_11011_01100_00000_000;
    5'b11100:nxt_state=23'b10101_10110_11000_00000_000;
    5'b11000:nxt_state=23'b10101_01101_10000_00000_000;
    5'b10000:nxt_state=23'b10101_01011_00000_00000_000;
    5'b00000:nxt_state=23'b10101_01010_00000_00000_000;
    5'b00001:nxt_state=23'b11010_10101_00000_00000_000;
    5'b00011:nxt_state=23'b11011_01010_10000_00000_000;
    5'b00111:nxt_state=23'b11011_01101_01000_00000_000;
    5'b01111:nxt_state=23'b11011_01101_10100_00000_000;
    default: nxt_state=23'b11111_11111_11111_11111_111;
endcase
endcase
end

```

解码模块的“解码模块” decoder_dec

```
//改变数码管应显示的字符(cur_char)和已经解码的个数(cur_cnt)和当前解码是否合法(ac)的状态机
```

```

always@(posedge clk_out or posedge r_out)
begin
    if(r_out)
        begin
            cur_char<=8'b11111111; //all close
            cur_cnt<=4'b0000;
            ac<=1'b0;
        end
    else
        cur_char<=nxt_char;
        cur_cnt<=nxt_cnt;
        ac<=nxt_ac;
end

```

```
//判断数码管显示是否已满，若已满则状态保持
//没满时，根据按键和传入的电码(code)，改变数码管显示个数、数码管应当显示的字符
```

```

always@*
begin
    if(cur_cnt==8)
        begin
            nxt_cnt=4'b1000;
            if(r_out) begin
                nxt_char=8'b1111_1111;
                nxt_cnt=4'b0000;
            end
        end
    else if(m_out) begin //press mid

```

```

        case({code_len,code})
            default: begin nxt_char=8'b11111111; nxt_cnt=cur_cnt; end//can't
decode
            code_0: begin nxt_char=8'b11000000 ;nxt_cnt=cur_cnt+1; end
            code_1:begin nxt_char=8'b11111001; nxt_cnt=cur_cnt+1;end
            code_2: begin nxt_char=8'b10100100; nxt_cnt=cur_cnt+1;end
            code_3:begin nxt_char=8'b10110000 ;nxt_cnt=cur_cnt+1; end
            code_4: begin nxt_char=8'b10011001;nxt_cnt=cur_cnt+1; end
            code_5:begin nxt_char= 8'b10010010; nxt_cnt=cur_cnt+1;end
            code_6:begin nxt_char=8'b10000010 ; nxt_cnt=cur_cnt+1;end
            code_7:begin nxt_char= 8'b11111000;nxt_cnt=cur_cnt+1; end
            code_8:begin nxt_char=8'b10000000; nxt_cnt=cur_cnt+1;end
            code_9:begin nxt_char=8'b10010000 ;nxt_cnt=cur_cnt+1; end
            a: begin nxt_char=8'b10001000;nxt_cnt=cur_cnt+1; end
            b:begin nxt_char= 8'b10000011;nxt_cnt=cur_cnt+1; end
            c:begin nxt_char= 8'b11000110; nxt_cnt=cur_cnt+1; end
            d:begin nxt_char= 8'b10100001; nxt_cnt=cur_cnt+1;end
            e:begin nxt_char= 8'b10000110; nxt_cnt=cur_cnt+1;end
            f:begin nxt_char= 8'b10001110 ; nxt_cnt=cur_cnt+1;end
            g:begin nxt_char=8'b1100_0010; nxt_cnt=cur_cnt+1;end
            h: begin nxt_char=8'b1000_1001; nxt_cnt=cur_cnt+1;end
            i:begin nxt_char= 8'b1111_0000 ; nxt_cnt=cur_cnt+1;end
            j:begin nxt_char=8'b1111_0001; nxt_cnt=cur_cnt+1;end
            k:begin nxt_char=8'b1000_1010; nxt_cnt=cur_cnt+1;end
            l:begin nxt_char=8'b1100_0111; nxt_cnt=cur_cnt+1;end
            m:begin nxt_char=8'b1100_1000; nxt_cnt=cur_cnt+1;end
            n:begin nxt_char= 8'b1010_1011; nxt_cnt=cur_cnt+1;end
            o:begin nxt_char=8'b10100011; nxt_cnt=cur_cnt+1;end
            p:begin nxt_char=8'b1000_1100 ; nxt_cnt=cur_cnt+1;end
            q:begin nxt_char= 8'b1001_1000; nxt_cnt=cur_cnt+1;end
            r:begin nxt_char= 8'b1100_1110; nxt_cnt=cur_cnt+1;end
            s:begin nxt_char= 8'b1011_0110; nxt_cnt=cur_cnt+1;end
            t:begin nxt_char=8'b1000_0111 ; nxt_cnt=cur_cnt+1;end
            u:begin nxt_char=8'b1100_0001; nxt_cnt=cur_cnt+1;end
            v:begin nxt_char=8'b1110_0011; nxt_cnt=cur_cnt+1;end
            w:begin nxt_char=8'b1000_0001; nxt_cnt=cur_cnt+1;end
            x_code:begin nxt_char=8'b1001_1011; nxt_cnt=cur_cnt+1;end
            y:begin nxt_char=8'b1001_0001; nxt_cnt=cur_cnt+1;end
            z_code:begin nxt_char= 8'b1010_0101 ; nxt_cnt=cur_cnt+1;end
        endcase
    end
    else if(r_out) begin
        nxt_char=8'b1111_1111;
        nxt_cnt=4'b0000;
    end
    else begin
        nxt_cnt=cur_cnt;
        nxt_char=cur_char;
    end
end

```

解码的数码管显示模块 decoder_tube

注：与编码的数码管显示共用seg_en代码

```
//8位分别需要显示的字符
//初始化为全不显示

reg [7:0] cd_1=8'b11111111;
reg [7:0] cd_2=8'b11111111;
reg [7:0] cd_3=8'b11111111;
reg [7:0] cd_4=8'b11111111;
reg [7:0] cd_5=8'b11111111;
reg [7:0] cd_6=8'b11111111;
reg [7:0] cd_7=8'b11111111;
reg [7:0] cd_8=8'b11111111;
```

```
//根据当前已经显示的个数和输入的字符给8位数码管赋值(char为来自dec模块的输出，即需要显示的字符)
```

```
always@(num_seg,char)
begin
  case(num_seg)
    4'b0000:begin cd_1=char; end
    4'b0001:begin cd_2=char; end
    4'b0010:begin cd_3=char;end
    4'b0011:begin cd_4=char;end
    4'b0100:begin cd_5=char; end
    4'b0101:begin cd_6=char; end
    4'b0110:begin cd_7=char;end
    4'b0111:begin cd_8=char; end
  endcase
end
```

编码部分

小键盘

```
//调用防抖模块debounce_up，对up键加以防抖

debounce_up u(clk, ~rst, up, up_out);
```

```
//对键盘时钟进行分频与计数，得到key_clk与cnt1
```

```
always @ (posedge clk, posedge rst)
  if(switch)
    begin
      cnt1<=1'b0;
    end
  else if (~rst)
    cnt1 <= 0;
  else
    cnt1 <= cnt1 + 1'b1;
  assign key_clk = cnt1[19]; //2^20/100M = 10ms
```

```
//记录按下按键的次数以及退格的次数
```

```
always @ (posedge key_pressed_flag, posedge rst) begin
  if (~rst)
```

```

    num_pressed <= 0;
else
    num_pressed <= num_pressed + 1'b1;
end

always @ (posedge up_out, posedge rst) begin
    if(switch) begin
        num_back<=0;
    end
    if (rst)
        num_back <= 0;
    else
        num_back <= num_back + 1'b1;
end

```

//扫描小键盘行列值，将每一次的按下的值存进keyboard_val数组里

```

//!声明keyboard_val数组，下标从1到8，下标是几代表第几次按的数(从1开始计数)
reg [3:0] keyboard_val [8:0];
always @ (posedge key_clk or posedge rst) begin
    if (rst) begin
        keyboard_val[1] <= 4'hA;
        keyboard_val[2] <= 4'hA;
        keyboard_val[3] <= 4'hA;
        keyboard_val[4] <= 4'hA;
        keyboard_val[5] <= 4'hA;
        keyboard_val[6] <= 4'hA;
        keyboard_val[7] <= 4'hA;
        keyboard_val[8] <= 4'hA;
    end
    else
        begin          //退格后归A
            case (num_seg)
                0:keyboard_val[1] <= 4'hA;
                1:keyboard_val[2] <= 4'hA;
                2:keyboard_val[3] <= 4'hA;
                3:keyboard_val[4] <= 4'hA;
                4:keyboard_val[5] <= 4'hA;
                5:keyboard_val[6] <= 4'hA;
                6:keyboard_val[7] <= 4'hA;
                7:keyboard_val[8] <= 4'hA;
            endcase
            if (key_pressed_flag)
                case ({col_val, row_val})
                    8'b1110_1110 : keyboard_val[num_seg] <= 4'h1;
                    8'b1110_1101 : keyboard_val[num_seg] <= 4'h4;
                    8'b1110_1011 : keyboard_val[num_seg] <= 4'h7;

                    8'b1101_1110 : keyboard_val[num_seg] <= 4'h2;
                    8'b1101_1101 : keyboard_val[num_seg] <= 4'h5;
                    8'b1101_1011 : keyboard_val[num_seg] <= 4'h8;
                    8'b1101_0111 : keyboard_val[num_seg] <= 4'h0;

                    8'b1011_1110 : keyboard_val[num_seg] <= 4'h3;
                    8'b1011_1101 : keyboard_val[num_seg] <= 4'h6;
                    8'b1011_1011 : keyboard_val[num_seg] <= 4'h9;

```

```

        endcase
    end
end

```

七段数码管

```

//对七段数码管进行分频

always @(posedge clk, negedge rst) // 分频为clk_seg
begin
    if(switch)
        begin
            cnt2 <= 0; //cnt归0
            clk_seg <= 0; //clk归0
        end
    else if (rst) // 复位
        begin
            cnt2 <= 0; //cnt归0
            clk_seg <= 0; //clk归0
        end
    else begin
        if (cnt2 == (period >> 1) - 1)
            begin
                clk_seg <= ~clk_seg; // clk取反, 两次取反为一个周期
                cnt2 <= 0; // cnt归0
            end
        else
            cnt2 <= cnt2 + 1; // cnt递增
    end
end

always@(posedge clk_seg, negedge rst) // 根据clk_seg改变scan_cnt
begin
    if (rst) // 复位
        scan_cnt <= 0; // scan_cnt归0
    else begin
        scan_cnt <= scan_cnt + 1; // scan_cnt递增
        if (scan_cnt == 3'd7) // scan_cnt一个周期完成
            scan_cnt <= 0; // scan_cnt归0
    end
end

```

//根据不同的num_seg与scan_cnt的变化，赋予seg_en不同的值

```

always @(switch,rst,scan_cnt,num_pressed, num_back) begin
    if(switch)
        num_seg = 0;
    else if (rst)
        num_seg = 0;
    else begin
        num_seg = num_pressed - num_back;
        //为避免核心代码冗长，此处以num_seg == 2做展示（省去num_seg为1~8其余代码）
        if(num_seg == 2) begin //要显示的数字个数为2
            case(scan_cnt)
                0: seg_en = 8'b0111_1111; //第0“时刻”只亮第1个灯
                1: seg_en = 8'b1011_1111; //第1时刻只亮第2个灯
            end
        end
    end
end

```

```

2: seg_en = 8'b1111_1111;           //其余时刻不亮
3: seg_en = 8'b1111_1111;
4: seg_en = 8'b1111_1111;
5: seg_en = 8'b1111_1111;
6: seg_en = 8'b1111_1111;
7: seg_en = 8'b1111_1111;
default: seg_en = 8'b1111_1111;
      endcase
    end
  end

```

//!开个numDisplay数组，下标从0到9，下标是几代表要表示哪一个数字

```

//打表写出所有数字表示方法
wire[7:0] numDisplay [9:0];
begin
  assign numDisplay[0] = 8'b1100_0000; // 0
  assign numDisplay[1] = 8'b1111_1001; // 1
  assign numDisplay[2] = 8'b1010_0100; // 2
  assign numDisplay[3] = 8'b1011_0000; // 3
  assign numDisplay[4] = 8'b1001_1001; // 4
  assign numDisplay[5] = 8'b1001_0010; // 5
  assign numDisplay[6] = 8'b1000_0010; // 6
  assign numDisplay[7] = 8'b1111_1000; // 7
  assign numDisplay[8] = 8'b1000_0000; // 8
  assign numDisplay[9] = 8'b1001_0000; // 9
end

```

//根据scan_cnt的改变，不同的时刻显示不同的数字

```

//!keyboard_val数组下标从1到8，下标是几代表第几次按的数(从1开始计数)
always @(scan_cnt) begin
  case(scan_cnt)
    // !在第i时刻显示第i次按的数
    0: seg_out = numDisplay[keyboard_val[1]];
    1: seg_out = numDisplay[keyboard_val[2]];
    2: seg_out = numDisplay[keyboard_val[3]];
    3: seg_out = numDisplay[keyboard_val[4]];
    4: seg_out = numDisplay[keyboard_val[5]];
    5: seg_out = numDisplay[keyboard_val[6]];
    6: seg_out = numDisplay[keyboard_val[7]];
    7: seg_out = numDisplay[keyboard_val[8]];
    default: seg_out = 8'b1111_1111;
  endcase
end

```

蜂鸣器

```

//分频计时，总共二十秒，在key_on为0时，令cnt固定为0
always@(posedge clk, posedge rst)//可能有问题
begin
  if(rst)
    begin
      cnt <= 1'b0;
    end

```

```

else if(key_on==0)
begin
    cnt<=1'b0;
end
else if(cnt != 35'd1_999_999_999)      //20s
    cnt <= cnt + 1'b1;

else
    cnt <= 1'b0;           //清空
end

```

```

/*
蜂鸣器播放核心代码，在读入key、modulee、key_on之后判断是否进入播放部分，计时器在对应的区间内，在每过一段时间对beep取反从而获取人耳能听到的声音
以下为播放key==1，模式为短模式时的代码
*/

```

```

always@(posedge clk)
if(key==1&&modulee==0&&key_on==1)
begin
    //第一个频带 (第一声 短)
    if(cnt < 35'd50_000_000)
        begin
            if(cnt%27'd49_999==0)
                beep <= ~beep;
        end
    //第二个频带 (第二声 长)
    else if(cnt > 35'd100_000_000 && cnt < 35'd200_000_000)
        begin
            if(cnt%27'd49_999==0)
                beep <= ~beep;
        end
    //第三个频带 (第三声 长)
    else if(cnt > 35'd250_000_000 && cnt < 35'd350_000_000)
        begin
            if(cnt%27'd49_999==0)
                beep <= ~beep;
        end
    //第四个频带 (第四声 长)
    else if(cnt > 35'd400_000_000 && cnt < 35'd500_000_000)
        begin
            if(cnt%27'd49_999==0)
                beep <= ~beep;
        end
    // (第五声 长)
    else if(cnt > 35'd550_000_000 && cnt < 35'd650_000_000)
        begin
            if(cnt%27'd49_999==0)
                beep <= ~beep;
        end
end

```

// 实现一键播放功能

```

else if(key_on_all)
begin
  if(cnt_buzzer>0&&cnt_buzzer<time1)
    begin
      key_on<=1;
      val<=keyboard_val[1];
    end
  else if(cnt_buzzer>time1&&cnt_buzzer<time1+t)
    begin
      key_on<=0;
    end
  else if(cnt_buzzer>time1+t&&cnt_buzzer<time1+t+time2)
    begin
      key_on<=1;
      val<=keyboard_val[2];
    end
  else if(time1+t+time2<cnt_buzzer&&time1+t*2+time2>cnt_buzzer)
    begin
      key_on<=0;
    end
  else if(time1+t*2+time2+time3<cnt_buzzer&&time1+t*2+time2+time3>cnt_buzzer)
    begin
      key_on<=1;
      val<=keyboard_val[3];
    end
  else
if(time1+t*2+time2+time3>cnt_buzzer)
begin
  key_on<=0;
end
else
if(time1+t*3+time2+time3<cnt_buzzer&&time1+t*3+time2+time3+time4>cnt_buzzer)
begin
  key_on<=1;
  val<=keyboard_val[4];
end
else
if(time1+t*3+time2+time3+time4<cnt_buzzer&&time1+t*4+time2+time3+time4>cnt_buzzer)
begin
  key_on<=0;
end
else
if(time1+t*4+time2+time3+time4<cnt_buzzer&&time1+t*4+time2+time3+time4+time5>cnt_buzzer)
begin
  key_on<=1;
  val<=keyboard_val[5];
end
else if(time1+t*4+time2+time3+time4+time5<cnt_buzzer&&time1+t*5+time2+time3+time4+time5>cnt_buzzer)
begin
  key_on<=0;
end
else if(time1+t*5+time2+time3+time4+time5<cnt_buzzer&&time1+t*5+time2+time3+time4+time5+time6>cnt_buzzer)
begin
  key_on<=1;

```

```

        val<=keyboard_val[6];
    end
else if(time1+t*5+time2+time3+time4+time5+time6<cnt_buzzer&&
        time1+t*6+time2+time3+time4+time5+time6>cnt_buzzer)
begin
    key_on<=0;
end
else if(time1+t*6+time2+time3+time4+time5+time6<cnt_buzzer&&
        time1+t*6+time2+time3+time4+time5+time6+time7>cnt_buzzer)
begin
    key_on<=1;
    val<=keyboard_val[7];
end
else if(time1+t*6+time2+time3+time4+time5+time6+time7<cnt_buzzer&&
        time1+t*7+time2+time3+time4+time5+time6+time7>cnt_buzzer)
begin
    key_on<=0;
end
else if(time1+t*7+time2+time3+time4+time5+time6+time7<cnt_buzzer&&
        time1+t*7+time2+time3+time4+time5+time6+time7+time8>cnt_buzzer)
begin
    key_on<=1;
    val<=keyboard_val[8];
end
else if(time1+t*7+time2+time3+time4+time5+time6+time7+time8<cnt_buzzer)
begin
    key_on<=0;
end
end
else key_on<=0;
end
end
buzzer1(clk,rst,modulee,key_on,val,beep); //此处为单数字播放功能

always@(posedge clk)
if(!switch)begin
begin
    case(keyboard_val[1])
        0:time1=TIME0_0;
        1:time1=TIME1_0;
        2:time1=TIME2_0;
        3:time1=TIME3_0;
        4:time1=TIME4_0;
        5:time1=TIME5_0;
        6:time1=TIME6_0;
        7:time1=TIME7_0;
        8:time1=TIME8_0;
        9:time1=TIME9_0;
    endcase
    case(keyboard_val[2])
        0:time2=TIME0_0;
        1:time2=TIME1_0;
        2:time2=TIME2_0;
        3:time2=TIME3_0;
        4:time2=TIME4_0;
        5:time2=TIME5_0;
        6:time2=TIME6_0;
        7:time2=TIME7_0;
    endcase
end

```

```

8:time2=TIME8_0;
9:time2=TIME9_0;
endcase
case(keyboard_val[3])
0:time3=TIME0_0;
1:time3=TIME1_0;
2:time3=TIME2_0;
3:time3=TIME3_0;
4:time3=TIME4_0;
5:time3=TIME5_0;
6:time3=TIME6_0;
7:time3=TIME7_0;
8:time3=TIME8_0;
9:time3=TIME9_0;
endcase
case(keyboard_val[4])
0:time4=TIME0_0;
1:time4=TIME1_0;
2:time4=TIME2_0;
3:time4=TIME3_0;
4:time4=TIME4_0;
5:time4=TIME5_0;
6:time4=TIME6_0;
7:time4=TIME7_0;
8:time4=TIME8_0;
9:time4=TIME9_0;
endcase
case(keyboard_val[5])
0:time5=TIME0_0;
1:time5=TIME1_0;
2:time5=TIME2_0;
3:time5=TIME3_0;
4:time5=TIME4_0;
5:time5=TIME5_0;
6:time5=TIME6_0;
7:time5=TIME7_0;
8:time5=TIME8_0;
9:time5=TIME9_0;
endcase
case(keyboard_val[6])
0:time6=TIME0_0;
1:time6=TIME1_0;
2:time6=TIME2_0;
3:time6=TIME3_0;
4:time6=TIME4_0;
5:time6=TIME5_0;
6:time6=TIME6_0;
7:time6=TIME7_0;
8:time6=TIME8_0;
9:time6=TIME9_0;
endcase
case(keyboard_val[7])
0:time7=TIME0_0;
1:time7=TIME1_0;
2:time7=TIME2_0;
3:time7=TIME3_0;
4:time7=TIME4_0;
5:time7=TIME5_0;

```

```

6:time7=TIME6_0;
7:time7=TIME7_0;
8:time7=TIME8_0;
9:time7=TIME9_0;
endcase
case(keyboard_val[8])
0:time8=TIME0_0;
1:time8=TIME1_0;
2:time8=TIME2_0;
3:time8=TIME3_0;
4:time8=TIME4_0;
5:time8=TIME5_0;
6:time8=TIME6_0;
7:time8=TIME7_0;
8:time8=TIME8_0;
9:time8=TIME9_0;
endcase
end

```

• 约束文件:

```

#encoder
set_property IOSTANDARD LVCMOS33 [get_ports {col[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {col[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {col[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {col[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {row[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {row[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {row[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {row[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_en[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {up}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports rst]

set_property IOSTANDARD LVCMOS33 [get_ports beep]
set_property IOSTANDARD LVCMOS33 [get_ports key_on1]
set_property IOSTANDARD LVCMOS33 [get_ports key_on2]
set_property IOSTANDARD LVCMOS33 [get_ports key_on3]
set_property IOSTANDARD LVCMOS33 [get_ports key_on4]
set_property IOSTANDARD LVCMOS33 [get_ports key_on5]
set_property IOSTANDARD LVCMOS33 [get_ports key_on6]

```

```

set_property IOSTANDARD LVCMOS33 [get_ports key_on7]
set_property IOSTANDARD LVCMOS33 [get_ports key_on8]
set_property IOSTANDARD LVCMOS33 [get_ports key_on_all]
set_property IOSTANDARD LVCMOS33 [get_ports modulee]
set_property IOSTANDARD LVCMOS33 [get_ports key_on_mus]

#decoder
set_property IOSTANDARD LVCMOS33 [get_ports {switch}]
set_property IOSTANDARD LVCMOS33 [get_ports {down}]
set_property IOSTANDARD LVCMOS33 [get_ports {short}]
set_property IOSTANDARD LVCMOS33 [get_ports {long}]
set_property IOSTANDARD LVCMOS33 [get_ports {mid}]

set_property IOSTANDARD LVCMOS33 [get_ports {led_warn}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[22]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[21]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[20]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[19]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[18]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[17]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[16]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {led_out[0]}]

#decoder led
set_property PACKAGE_PIN K17 [get_ports {led_out[22]}]
set_property PACKAGE_PIN L13 [get_ports {led_out[21]}]
set_property PACKAGE_PIN M13 [get_ports {led_out[20]}]
set_property PACKAGE_PIN K14 [get_ports {led_out[19]}]
set_property PACKAGE_PIN K13 [get_ports {led_out[18]}]
set_property PACKAGE_PIN M20 [get_ports {led_out[17]}]
set_property PACKAGE_PIN N20 [get_ports {led_out[16]}]
set_property PACKAGE_PIN N19 [get_ports {led_out[15]}]
set_property PACKAGE_PIN M17 [get_ports {led_out[14]}]
set_property PACKAGE_PIN M16 [get_ports {led_out[13]}]
set_property PACKAGE_PIN M15 [get_ports {led_out[12]}]
set_property PACKAGE_PIN K16 [get_ports {led_out[11]}]
set_property PACKAGE_PIN L16 [get_ports {led_out[10]}]
set_property PACKAGE_PIN L15 [get_ports {led_out[9]}]
set_property PACKAGE_PIN L14 [get_ports {led_out[8]}]
set_property PACKAGE_PIN J17 [get_ports {led_out[7]}]
set_property PACKAGE_PIN F21 [get_ports {led_out[6]}]
set_property PACKAGE_PIN G22 [get_ports {led_out[5]}]
set_property PACKAGE_PIN G21 [get_ports {led_out[4]}]

```

```

set_property PACKAGE_PIN D21 [get_ports {led_out[3]}]
set_property PACKAGE_PIN E21 [get_ports {led_out[2]}]
set_property PACKAGE_PIN D22 [get_ports {led_out[1]}]
set_property PACKAGE_PIN E22 [get_ports {led_out[0]}]
set_property PACKAGE_PIN A21 [get_ports {led_warn}]

#decoder button
set_property PACKAGE_PIN Y9 [get_ports {switch}]
set_property PACKAGE_PIN P2 [get_ports {down}]
set_property PACKAGE_PIN P1 [get_ports {short}]
set_property PACKAGE_PIN R1 [get_ports {long}]
set_property PACKAGE_PIN P4 [get_ports {mid}]

#encoder keypad
set_property PACKAGE_PIN M2 [get_ports {col[3]}]
set_property PACKAGE_PIN K6 [get_ports {col[2]}]
set_property PACKAGE_PIN J6 [get_ports {col[1]}]
set_property PACKAGE_PIN L5 [get_ports {col[0]}]

set_property PACKAGE_PIN K4 [get_ports {row[3]}]
set_property PACKAGE_PIN J4 [get_ports {row[2]}]
set_property PACKAGE_PIN L3 [get_ports {row[1]}]
set_property PACKAGE_PIN K3 [get_ports {row[0]}]

set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets up]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets clk]
set_property CLOCK_DEDICATED_ROUTE FALSE [get_nets switch]

#encoder 7segtube
set_property PACKAGE_PIN F15 [get_ports {seg_out[0]}]
set_property PACKAGE_PIN F13 [get_ports {seg_out[1]}]
set_property PACKAGE_PIN F14 [get_ports {seg_out[2]}]
set_property PACKAGE_PIN F16 [get_ports {seg_out[3]}]
set_property PACKAGE_PIN E17 [get_ports {seg_out[4]}]
set_property PACKAGE_PIN C14 [get_ports {seg_out[5]}]
set_property PACKAGE_PIN C15 [get_ports {seg_out[6]}]
set_property PACKAGE_PIN E13 [get_ports {seg_out[7]}]
set_property PACKAGE_PIN C19 [get_ports {seg_en[0]}]
set_property PACKAGE_PIN E19 [get_ports {seg_en[1]}]
set_property PACKAGE_PIN D19 [get_ports {seg_en[2]}]
set_property PACKAGE_PIN F18 [get_ports {seg_en[3]}]
set_property PACKAGE_PIN E18 [get_ports {seg_en[4]}]
set_property PACKAGE_PIN B20 [get_ports {seg_en[5]}]
set_property PACKAGE_PIN A20 [get_ports {seg_en[6]}]
set_property PACKAGE_PIN A18 [get_ports {seg_en[7]}]

#buzzer
set_property PACKAGE_PIN P5 [get_ports {up}]
set_property PACKAGE_PIN U6 [get_ports key_on1]
set_property PACKAGE_PIN W5 [get_ports key_on2]
set_property PACKAGE_PIN W6 [get_ports key_on3]
set_property PACKAGE_PIN U5 [get_ports key_on4]
set_property PACKAGE_PIN T5 [get_ports key_on5]
set_property PACKAGE_PIN T4 [get_ports key_on6]
set_property PACKAGE_PIN R4 [get_ports key_on7]
set_property PACKAGE_PIN W4 [get_ports key_on8]
set_property PACKAGE_PIN V5 [get_ports key_on_all]
set_property PACKAGE_PIN R6 [get_ports modulee]

```

```
set_property PACKAGE_PIN T6 [get_ports key_on_mus]  
  
set_property PACKAGE_PIN Y18 [get_ports clk]  
set_property PACKAGE_PIN P20 [get_ports rst]  
set_property PACKAGE_PIN A19 [get_ports beep]
```

总结&优化

1. 问题与解决方案

- 问题：在设计数码管的显示时遇到了较多bug，包括但不限于：显示错误字符、无法显示不同字符、无法控制字符一位一位显示出来、后一位字符会改变前一位字符等等
- 解决方案：由于数码管和解码模块的代码并不是单人负责的，因此我积极与负责数码管模块的队友交流，共同检查代码中的逻辑问题，包括*通配符导致的不稳定、检查分频器是否正确工作、利用testbench检查输入输出是否出现了问题。每当我们对代码做一次修改，都会将原代码备份，防止越改越乱，提高解决效率。
- 问题：在总合成top模块时，由于在解码和编码模块中都用到了数码管输出，导致出现了经典的multi-driver问题。
- 解决方案：我们在top模块中设计了一个复用器multiplexer，用一个拨码开关选择数码管最终的输出，有效解决了multi-driver问题。
- 问题：在总合成top模块后，发现编码模式对应的重置按键失效，将数码管复位成全亮状态。
- 解决方案：由于单独的编码模块是完全有效的，因此我们考虑着重检查合成过程中加入的代码出现了bug。经过反复对比前后版本代码，我们发现这一bug源自编码模式的退格功能使能信号的判断。删除对应代码后，此bug被有效解决。
- 问题：在总合成top模块后，发现状态指示灯的加入会导致解码模块的数码管出现显示错误。
- 解决方案：我们最初指示灯的设计是直接在解码模块的LED输入中多加入一个始终状态为1'b1的LED灯，但测试发现此修改会导致数码管出现问题。经过思考，我们决定用逆向思维，在顶层模块的multiplexer中，对顶层的LED灯输出进行选择，即在编码模块中也加入LED灯的输出，并保持解码模块不变。运用此设计，我们有效解决了这个bug。
- 问题：退格后若打开已退格过的蜂鸣器开关，蜂鸣器仍会播放
- 解决方案：将退格后的keyboard_val赋值为蜂鸣器无法对应的值，则不会播放
- 问题：一个always里面除rst信号外无法内置多个posedge，数码管的退格功能难以实现
- 解决方案：创造两个变量num_pressed以及num_back，二者相减得到num_seg
- 问题：如何存下多次按下的keyboard_val
- 解决方案：开个下标从1-8的数组即可存储
- 问题：如何指定数码管亮的个数

- 解决方案：传递变量num_seg，根据不同的num_seg对seg_en分别指定
- 问题：如何将小键盘按下去的数字与该显示的数联系起来
 - 解决方案：先对numDisplay数组每一个数字做初始化，后套二层数组: seg_out = numDisplay[keyboard_val[i]]即可
- 问题：LED灯会在1秒后自动熄灭
 - 解决方案：经过检查状态机的代码，发现当时钟上升沿到来时，若没有检测到按键输入，未设计停留在自身状态的代码。
- 问题：蜂鸣器一开始响不了
 - 解决方案：降低beep取反的速度，从而获得人耳能听到的频率
- 问题：与队友代码合成时，使用多个buzzer模块导致输出混乱错误
 - 解决方案：只保留一个模块，通过always语句将需要的key值和key_on导入模块中
- 问题：一键播放时，无法正常播放出需要的声音
 - 解决方案：在buzzer的上一层模块中加入时序逻辑和对应的播放时间。
- 问题：新增音乐功能后，解码时出现七段数码管两两同化现象
 - 解决方案：重新调整解码分频器后成功

2. 系统特色

- 编码模块设计清晰明了，对小键盘和数码管分别分频，利用数组巧妙将小键盘按下去的数值传给数码管区。小键盘模块即按即用，内置防抖功能，毫无操作难度，用户体验极佳。同时本模块内置了退格以及重置清空数码管等功能，方便用户重复多次使用本模块。
- 解码模块设计以用户体验为重：使用5个按键输入（短键、长键、解码键、退格键、重置键），每个按键都配有防抖功能，用户也可以通过按住单一按键实现连续输入，易于操作；使用LED实时显示已经输入的长短码，若用户失误输入错误的长短码，有退格按钮供用户还原，提供无法解码报警和数码管已满报警功能，便于用户使用；每解码一次都会将解码的结果保留在七段数码管中，十分清晰，也有配套的重置键清空数码管，便于用户重复使用。
- 解码模块内部设计逻辑清晰易懂：解码模块分为四个小模块：，每个模块都有对应的状态机保存当前状态。以输入模块为例，状态包括当前输入位数长度和当前输入长短码类别，每次检测到相应按键，便更新状态到下一个状态，若未检测到按键，则保留当前状态。每个模块的输出都作为下一个模块的输入，即输入模块输出一个8位的比特串，高3位代表当前电码长度，低5位代表长短码种类（0代表短，1代表长），这一比特串被传到LED模块和解码模块中，LED模块输出对应的LED显示，解码模块根据当前比特串输出对应数码管的输出，传入到数码管显示模块中。
- 蜂鸣器播放模块（buzzer）的特色是该模块只接收单个位置的值（key）、是否播放该位置（key_on）以及模式（modulee），将该模块作为一种函数将数码管显示模块（keyandseg）中的key导入进行播放。buzzer作为数码管显示模块的子模块，keyandseg将每个格子对应的数字和开关是否打开导入到buzzer模块中，播放对应的key的声音，而一键播放则是在keyandseg上加入

每个数字对应的播放时间，然后在时序逻辑中给每一个数码管位置附上time的值，最后通过时序逻辑再播放出来。

- 使用模块化编程，逻辑清晰，易于debug：输入、解码、LED显示、数码管显示四个模块被一个解码顶层模块接线，实现了清晰的模块化编程。这样做好处是，可以编写独立的testbench测试对应模块的功能，精准定位bug出现的位置，并且也使整体逻辑更加清晰，有效提高了开发的效率。

3. 优化方向

- 本解码模块使用了大量枚举，将总共36种可能出现的有效电码可能性以枚举的形式体现在设计中（LED显示、解码、数码管显示），使得部分代码显得冗长，并且增加了重复劳动的工作量。日后可以优化这部分逻辑，使用更高效简洁的代码实现这部分功能。
- 本解码模块并未对分频器进行进一步测试优化，由于设计是在时钟上升沿检测是否有按键被按下，如果用户按键时长过短或过长，可能导致未读取到输入，或者错误读取到了多次输入的问题。日后可以对分频器进一步优化，以实现更加用户友好的操作。
- 解码模块中部分代码使用了过于频繁的if-else逻辑，日后可以考虑将if-else改为case逻辑，以增加系统的工作效率和稳定性。
- 编码模块未对极致的模块化编程进行执行。在作业中仅仅为了显示一串数字，便分了三个module来实现。而在编码的keyANDseg模块中，集成了小键盘，数码管功能。由于小键盘和数码管写成的时间较早，在当时未形成完整的模块化编程思维，给后续的debug也造成了许多麻烦。在后续的计算机组成原理，嵌入式等课程中，一定要对Verilog此类硬件描述语言重要的思想——模块化编程牢记在心。
- 播放的声音有时候会比较刺耳、断断续续，可以考虑通过改变频率来让声音悦耳。一键播放和模式切换的代码实现比较臃肿，风格不太统一，比较偏向于打表实现，可以设置几个代表长短播放的参数，能够实现更多的模式切换，也能够让结构更加清晰。