数字系统设计实验报告

选号机的设计与实现

学院：信息与通信工程学院

班级：2017211128

姓名：郑宇博

学号：2017210103

目录

一、设计课题的任务要求

题目　**选号机的设计与实现**

设计一个选号机，可以选择以一位汉字、一位字母和五位阿拉伯数字组成一串号码。

（一）基本要求：

1、用 SW7 作为选号机开关，打开开关 SW7 后选号机自检：8*8 点阵和数码管 DISP7~DISP0 全亮 0.5S 熄灭 0.5S 重复三次，进入待机状态；

2、 使用按键 BTN7 进入选号状态，按以下顺序进行选号，当前面的号码未选定时，后 面的按键无效。具体要求如下：

a) 8*8 点阵轮流显示“京”“ 沪”“ 吉”“ 苏”“ 川”五个汉字，每个汉字显示停留 时间 0.5S，按动 BTN6 选中当前显示的汉字，该汉字稳定显示；

b) 数码管 DISP5 上轮流显示“A”“ C”“ E”“ F”“ H”“ L”六个大写字母，每个字 母显示停留时间 0.4S，按动 BTN5 选中当前显示的字母，该字母稳定显示；

c) 数码管 DISP4 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.3S，按 动 BTN4 选中当前显示的数字，该数字稳定显示；

d) 数码管 DISP3 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.2S，按 动 BTN3 选中当前显示的数字，该数字稳定显示；

e) 数码管 DISP2 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.1S，按 动 BTN2 选中当前显示的数字，该数字稳定显示；

f) 数码管 DISP1 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.08S，按 动 BTN1 选中当前显示的数字，该数字稳定显示；

g) 数码管 DISP0 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.05S，按 动 BTN0 选中当前显示的数字，该数字稳定显示；

3、 DISP0 内容选定后表示所有内容选择完毕，所有内容整体以 2Hz 闪烁三次以示提醒， 然后稳定显示；

4、 使用按键 BTN7 可以重新进入选号状态，再一次进行选号。

（二）提高要求：

1、 自检过程、各项内容滚动时、内容选定后进行闪烁提醒时伴有适当的音乐，各个按 键按下时伴有按键音；

2、 各个数码管显示数字的方式改为随机显示“0~9”十个数字中的一个；

3、 自拟其他功能。

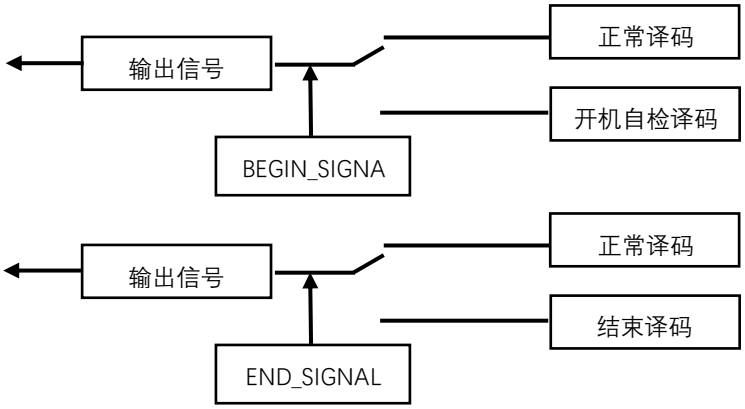
二、系统设计

（一）设计思路

1. 设计底层模块：分频模块，按键消抖模块，点阵解码模块，数码管解码模块，检测按键模块，开机自检模块，完成闪烁模块和按键声音模块

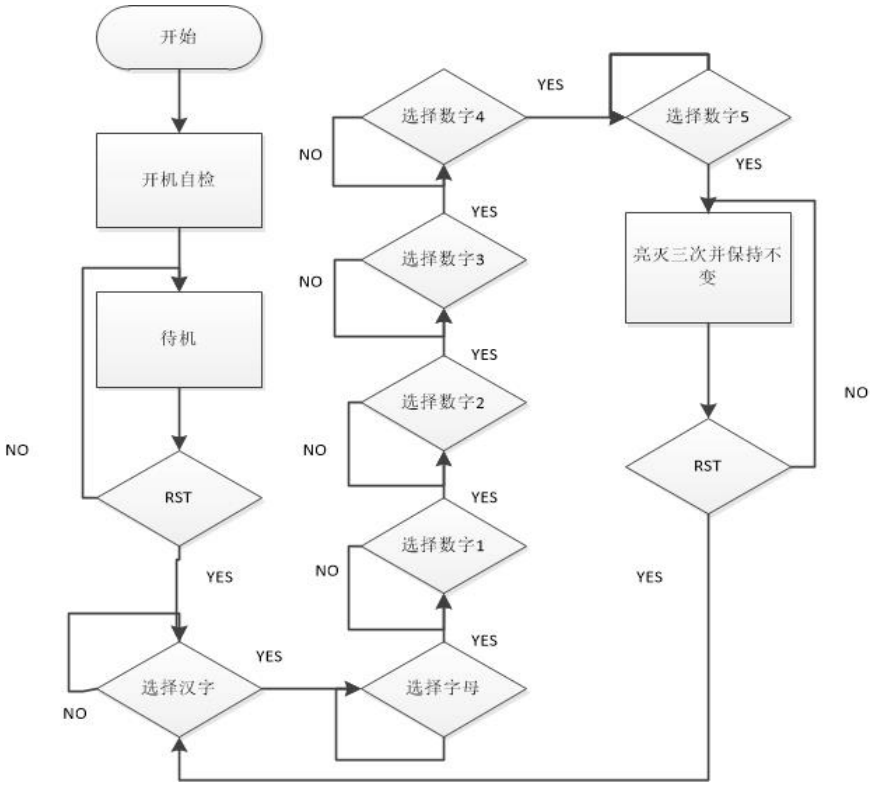
2. 设计顶层模块：在完成案件检测模块之后开始设计顶层模块，加入开机自检和闪烁，最后加入按键声音。

整体分为 3 个状态。状态机需要设置两个“单刀双掷开关”，在开机处有一个，在结束处为另一个。开机时开关 1 在 1 位置，进行开机自检操作；按下复位键之后开关到 2 位置，进行后续状态操作。结束时开关 2 本来打在 3 位置，为保持之前的显示，一旦结束，开关 2 打到 4 位置，在 4 位置执行结束亮灭显示和持续显示以及复位循环的操作。

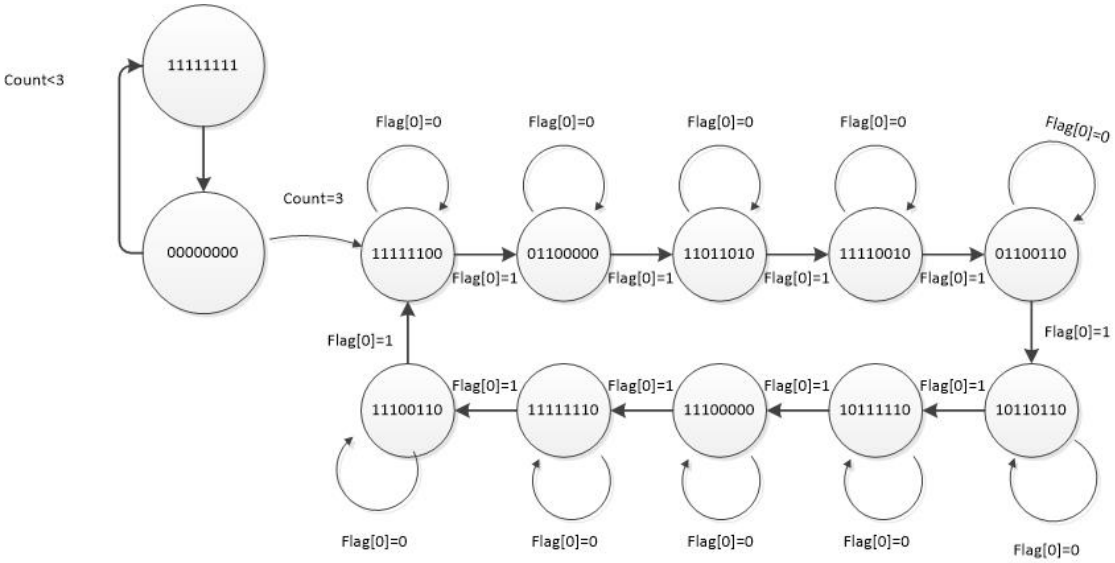


（二）总体框图

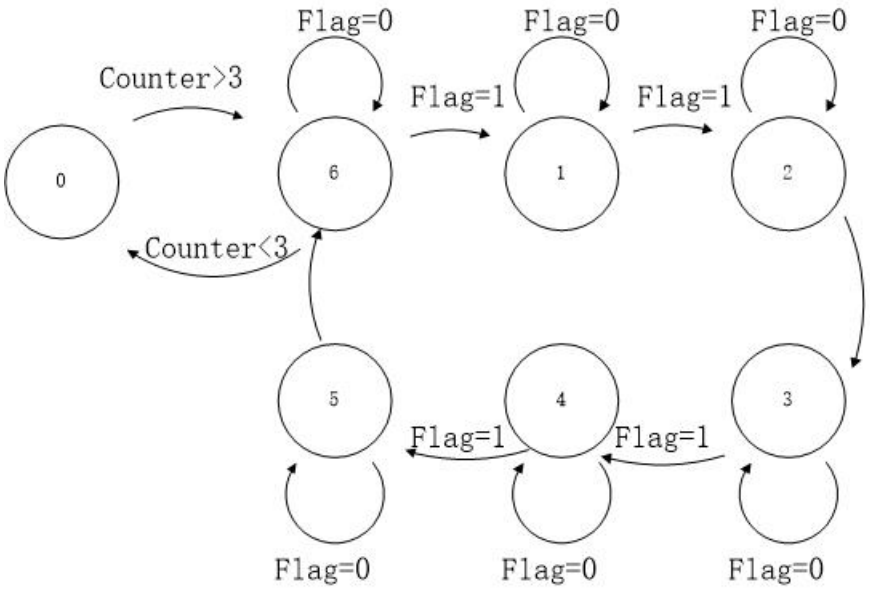
1. 流程图



2. 状态转移图



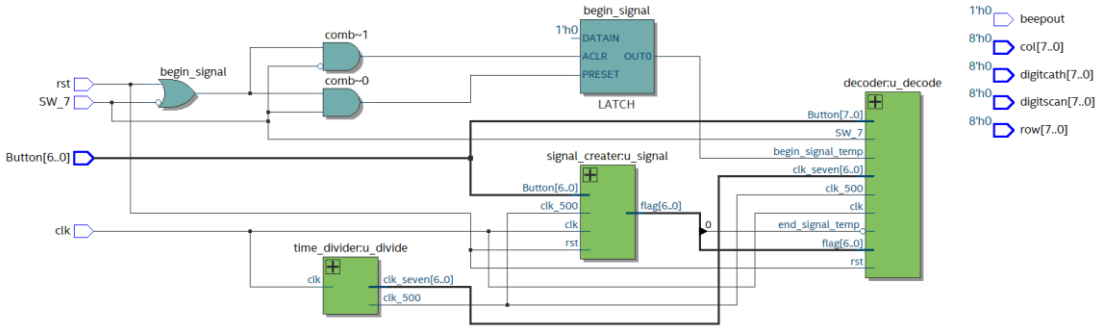
单个数码管状态转移图

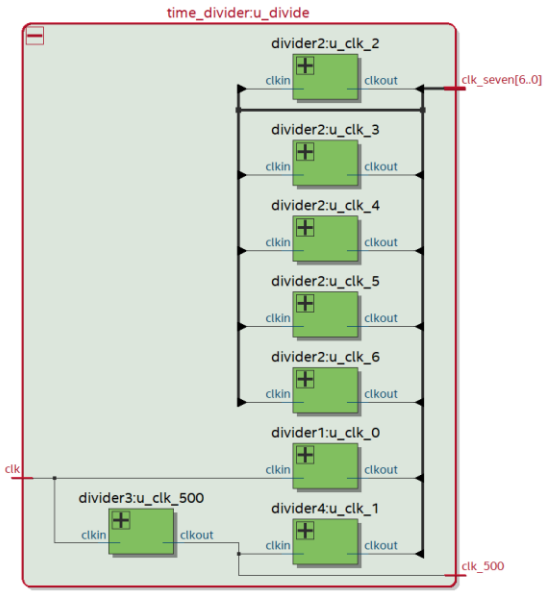


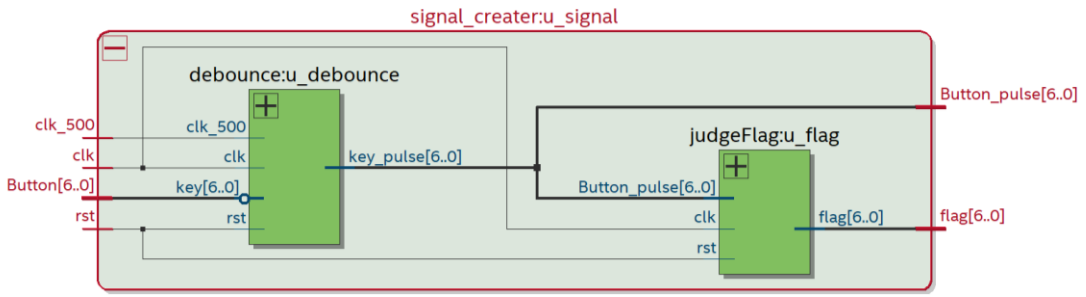
1:00010000_11111111_01111110_01000010_01111110_01010100_11010010_00110000
2:11000100_01011111_10010001_00011111_01010001_01010000_10100000_10100000
3:00011000_11111111_00011000_01111110_00000000_01111110_01000010_01111110
4:00100100_11111111_00010000_11111100_01010110_10100101_01100100_11001100
5:01001001_01001001_01001001_01001001_01001001_01001001_01001001_10001001
6:11111111_11111111_11111111_11111111_11111111_11111111_11111111_11111111
0:00000000_00000000_00000000_00000000_00000000_00000000_00000000_00000000

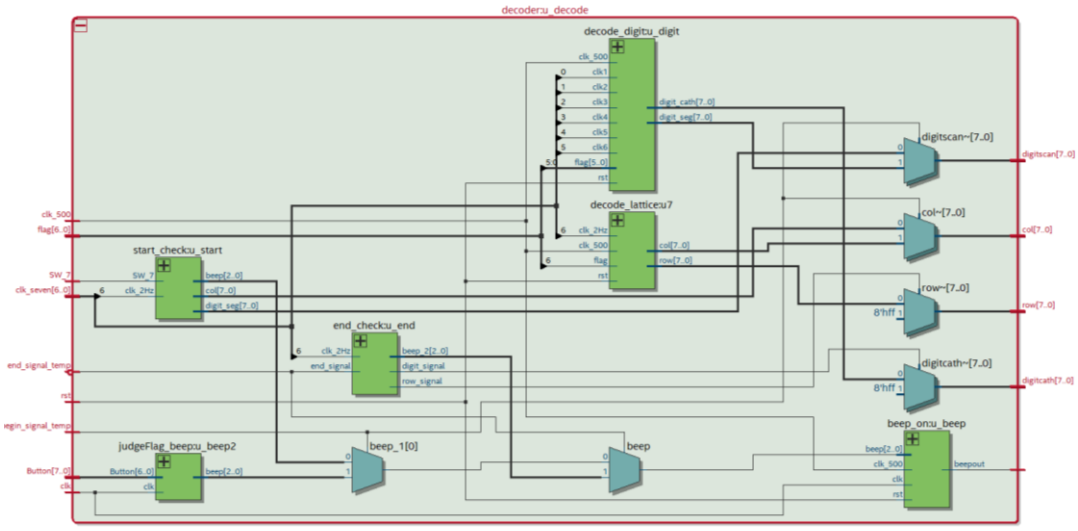
点阵状态转移图

3. RTL 图


begin_signal
1'h0
DATAIN
ACLR
OUT0
PRESET
LATCH
comb~1
comb~0
begin_signal
rst
SW_7
Button[6..0]
clk
signal_creater:u_signal
Button[6..0]
clk_500
clk
rst
flag[6..0]
time_divider:u_divide
clk
clk_seven[6..0]
clk_500
decoder:u_decode
Button[7..0]
SW_7
begin_signal_temp
clk_seven[6..0]
clk_500
clk
end_signal_temp
flag[6..0]
rst
1'h0
beepout
8'h0
col[7..0]
8'h0
digitcath[7..0]
8'h0
digitscan[7..0]
8'h0
row[7..0]


time_divider:u_divide
divider2:u_clk_2
clkin
clkout
clk_seven[6..0]
divider2:u_clk_3
divider2:u_clk_4
divider2:u_clk_5
divider2:u_clk_6
divider1:u_clk_0
clk
divider3:u_clk_500
divider4:u_clk_1
clk_500


signal_creater:u_signal
debounce:u_debounce
clk_500
clk
Button[6..0]
rst
key[6..0]
key_pulse[6..0]
judgeFlag:u_flag
Button_pulse[6..0]
clk
rst
flag[6..0]
Button_pulse[6..0]
flag[6..0]



（三）分块设计

1. 分频器设计

分频器是指使输出信号频率为输入信号频率整数分之一的电子电路，为了节省资源，我的项目使用串行分频。先将 50MHz 的系统频率分成 500Hz 时钟和 20Hz 时钟各一个，再由 500Hz 时钟分出 12.5Hz 时钟，由 20Hz 时钟分出 10Hz、5Hz、2.5Hz、2Hz 的时钟。

分频器为后面所有功能提供时钟信号。

2. 信号发生器设计

信号发生器由两部分组成：*按键消抖模块和按键统计模块*。

在机械按键的触点闭合和断开时，都会产生抖动，为了保证系统能正确识别按键的开关，就必须对按键的抖动进行处理。系统检测到按键按下动作之后进行 10ms~20ms 左右的延时，当前沿的抖动消失之后再一次检测按键的状态。如果仍然是按下的电平状态，则认为这是一次真正的按键按下；同样检测到按键释放，也要做 10ms~20ms 延时，检测到后沿抖动消失后认为是一个完整的按键弹起过程。此处我使用了脉冲边沿检测的方法：用一个频率更高的时钟去触发要检测的信号，用两个寄存器去储存相邻两个时钟采集到的值，然后进行异或运算，如果不为零，代表发生了上升沿或者下降沿。

按键统计模块是为了实现后面的不可跳跃选择功能而设立的，类似于使能信号，只有当前一个按键对应的使能信号有效下当前按键有效时，当前按键的使能端也才会有效。

信号发生器输出为 7 个消抖完成的 KEY_PULSE 和 7 个使能信号 FLAG。

3. 译码器设计

译码器分为五个部分：*开机自检译码，结束译码，点阵译码，数码管译码，蜂鸣器译码。*

开机自检译码只负责输出一种点阵的列信号和数码管段选信号，若单刀双掷开关（RST）打到开机，则输出的是开机自检模块制造的列信号和数码管段选信号。利用了一个频率为 2Hz 的 6 位计数器来控制输出。

结束译码只负责输出一种点阵的行扫描信号和数码管位选端信号。如果结束信号传来，则输出的行扫描信号和数码管位选端信号根据结束译码制造的信号输出。利用了一个频率为 2Hz 的 6 位计数器来控制输出。

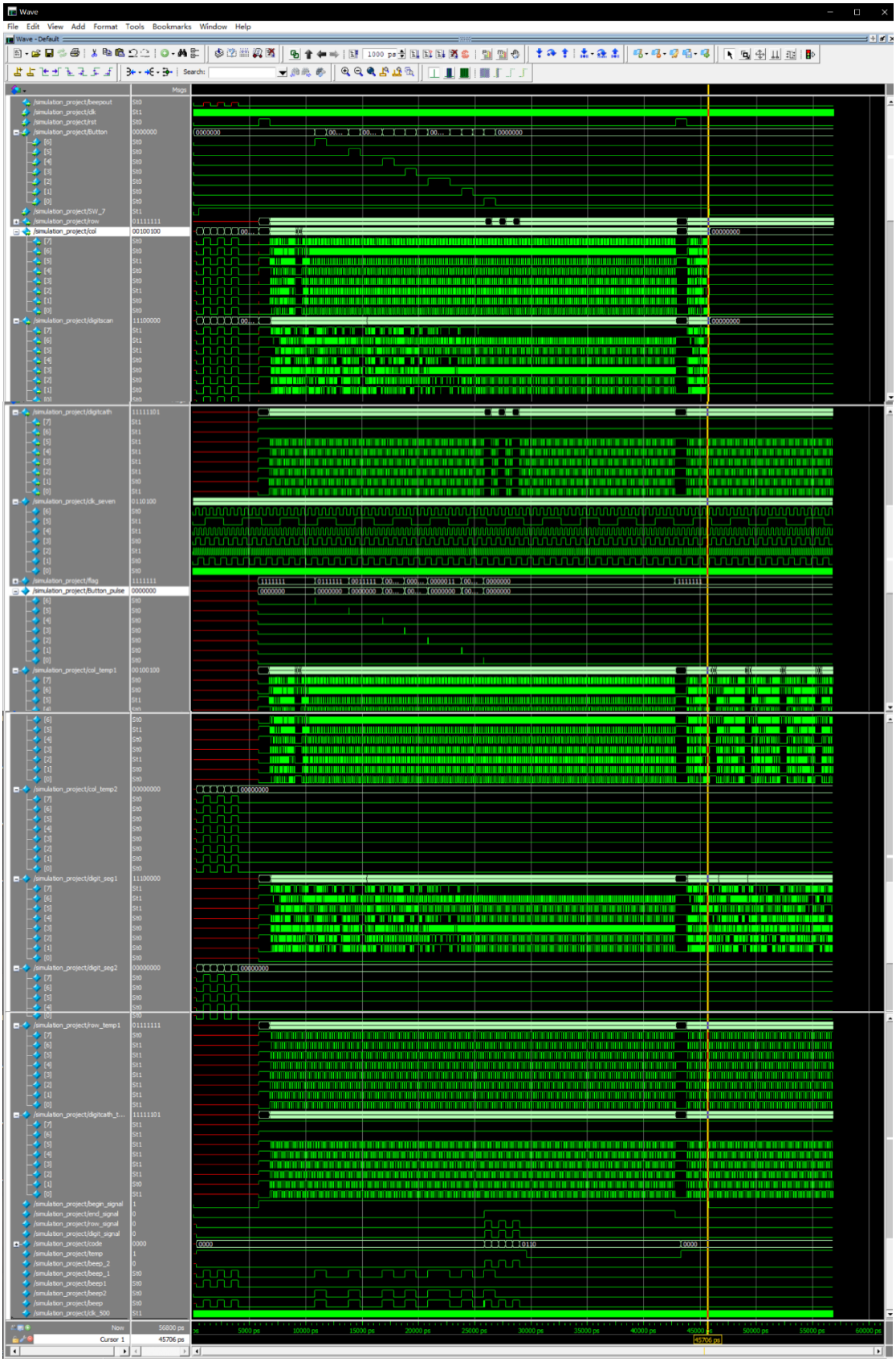
点阵译码器负责输出循环信号产生的点阵图像，同时由**信号发生器**传来的 flag 信号判定是否保持输出不变。利用了一个频率为 2Hz 的 5 位计数器和一个频率为 500Hz 的 8 位的计数器来进行行扫描和列赋值。

数码管译码器负责输出循环信号产生的数码管图像，同时由**信号发生器**传来的 flag 信号判定是否保持输出不变。此处利用了 6 个不同频率（来自**分频器**）的 4 位计数器，来控制不同数码管的更换数字频率，同时使用一个频率为 500Hz 的 6 位计数器来控制数码管的位选输出信号。

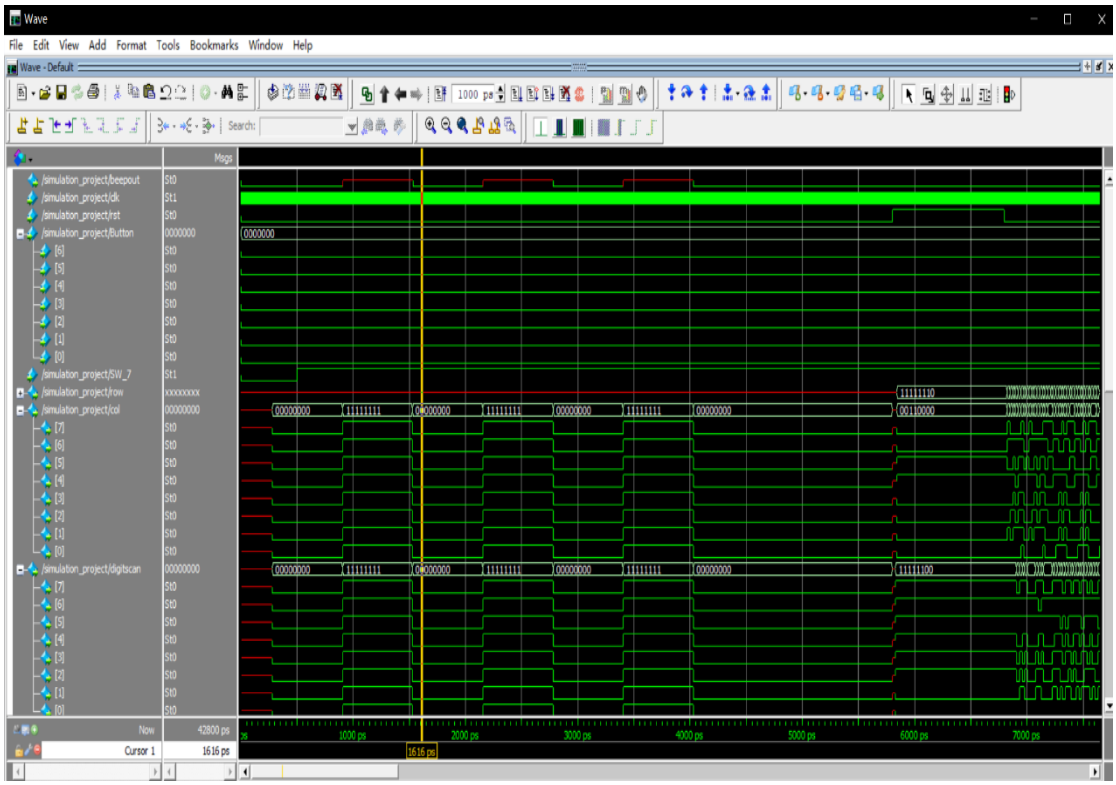
蜂鸣器译码负责输出不同频率的音频信号，它的使能来自**信号发生器**，同时在这个模块中我设置了 6 个分频器来制造 6 个音阶，再引入来自**分频器**的 500Hz 频率，便制造了 7 个音阶，在不同按键有效时或者开机关机时放出不同的声音信号。

三、仿真波形及波形分析

（一）全过程仿真（顶层模块仿真）

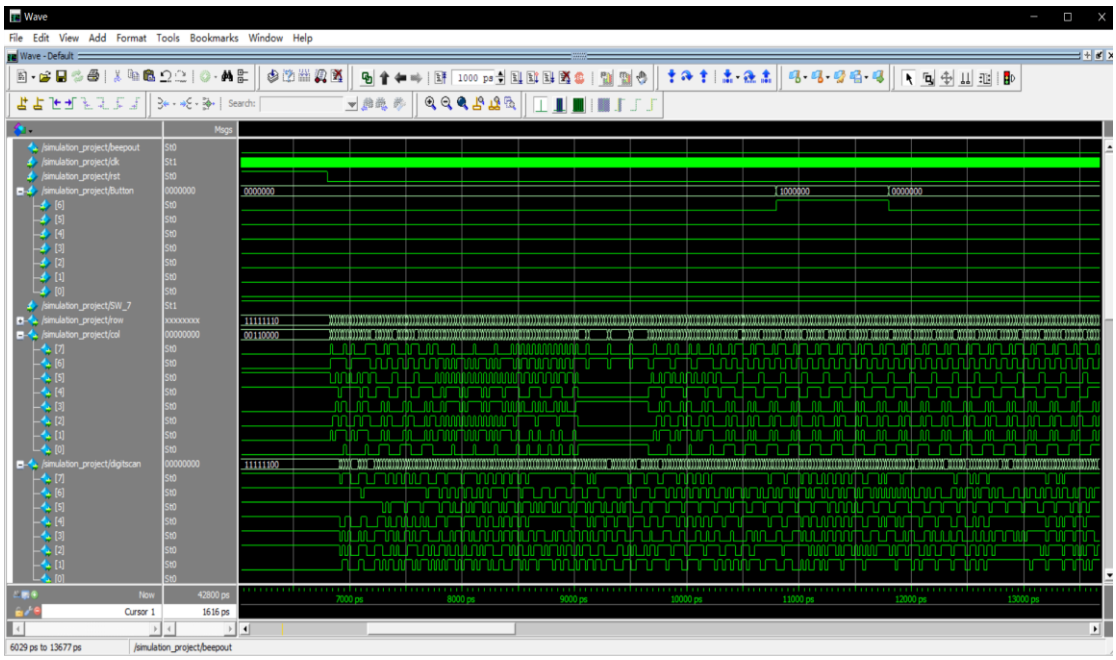


开机自检部分



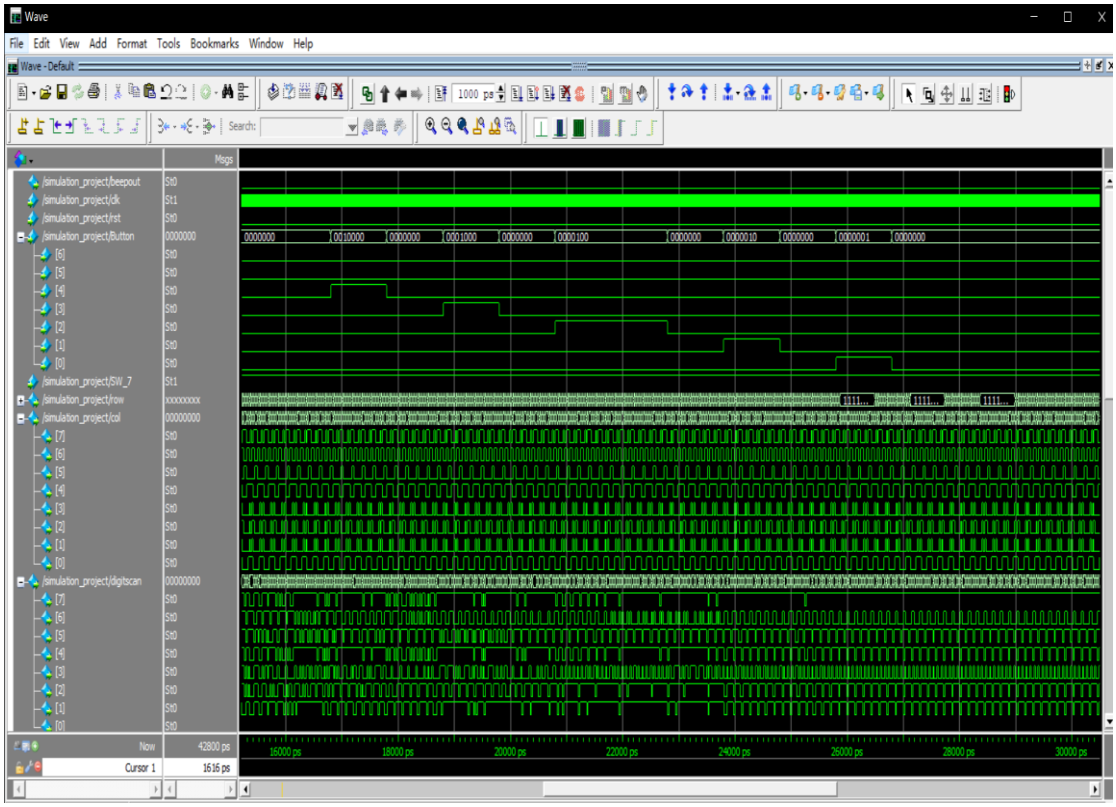
开机自检内容如上图所示，经过几个时间周期之后，将 SW_7 拨码开关的值变为 1，等待下一个时间周期的上升沿到来时开始自检，所有点阵和晶体管亮灭三次后进入全 0 的待机状态，等待 rst 按键的到来进行循环扫描显示内容

选汉字部分



RST 信号到来之后，开始循环扫描，给 Button[6]一个高电平信号，则点阵对应的汉字的扫秒停止循环，保持选好的汉字显示，而数码管的扫描还在继续。

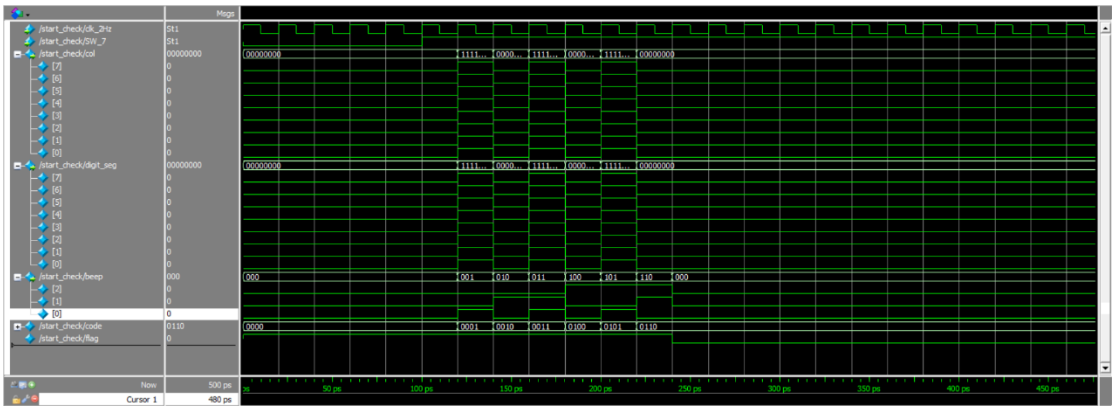
选字母，数字和结束部分



由上图可以看到，依次给 button[5]到 Button[0]一个高电平信号，数码管的段选渐渐变为恒定的值，直到按下 Button[0],数码管的段选彻底固定，并进入结束阶段。在这个过程中，如果在 Button[m]还没有给过高电平的时候，提前按下了 Button[n],(m>n),则 Button[n]对应的数码管并不会停止显示。这与另一个模块 JUDGE_FLAG 的相关功能有管，其仿真请看第 10 页。结束后系统有三个亮灭周期，完成后显示选好的所有号码。

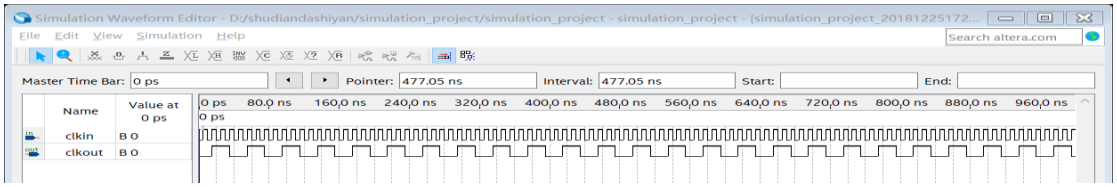
（二）分模块仿真

1. 开机自检模块仿真



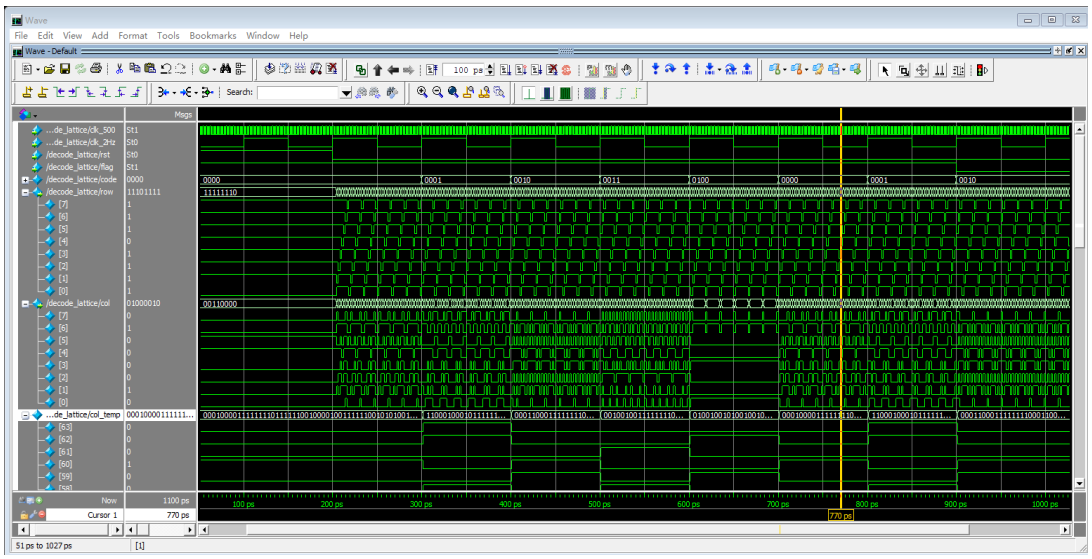
开机后即 SW_7 赋值为 1 后，系统在下一个时钟上升沿开始自检，对列和段选的控制为 1->0->1->0->1->0，然后保持。

2. 分频器模块仿真



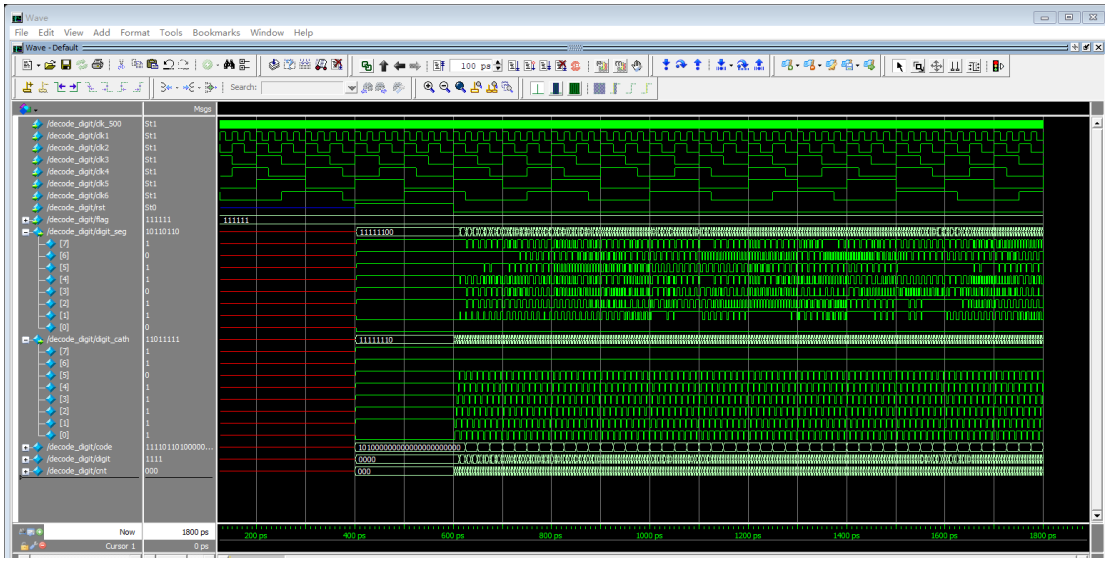
此仿真直接使用 Quartus 自带的 Simulation Waveform Editor 进行，模拟的是四分频的分频器，至于我的项目里其他分频器的仿真占用篇幅较多就不一一说明。

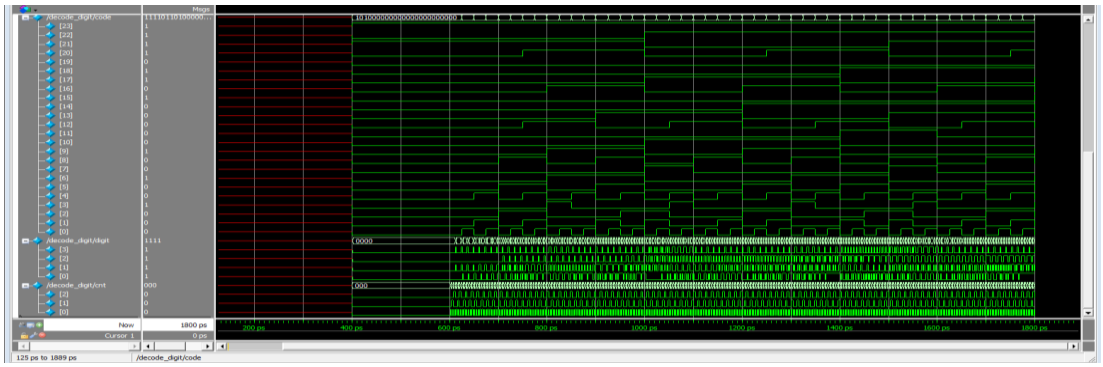
3. 点阵仿真



由上图可见，点阵的输出为行扫描和列赋值的结合。在循环扫描过程中，计数器模为 5，每五个计数周期循环一次。

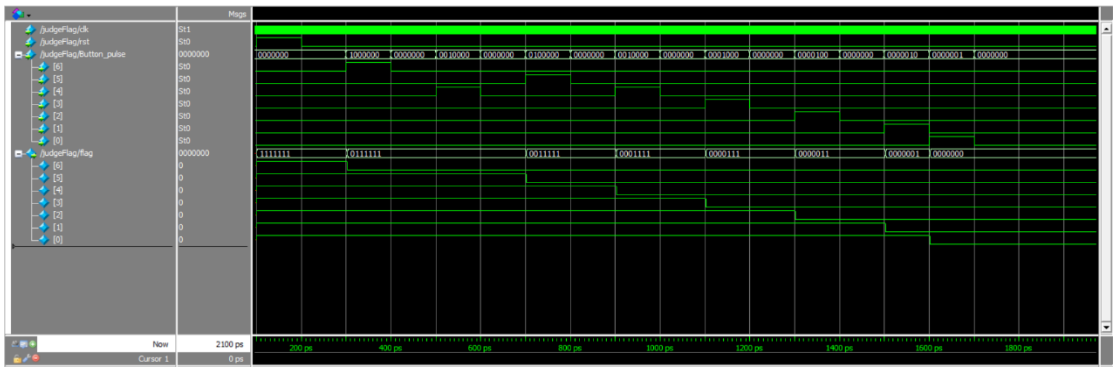
4. 数码管仿真





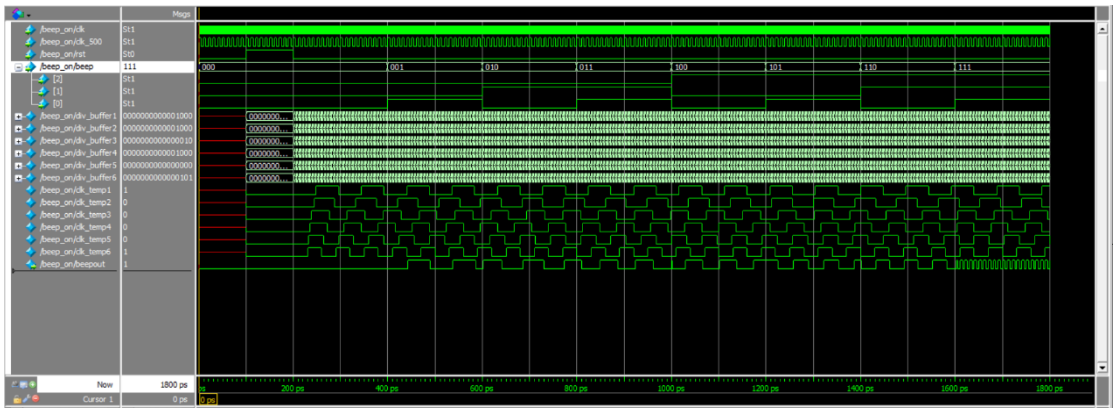
数码管为段选和位选同时进行，而不同位的变化频率不同，造成仿真波形中有的段较密，有的段较疏。在数码管的解码过程中，24 位计数器以不同的时间信号进行计数控制不同的位的频率，可以从第二张图看出。

5. 检测按键仿真



每一个消抖过的按键对应一个 flag，当按照 Button [6]->Button[0]的顺序依次赋一段时间高电平时，flag[6]->flag[0]也会依次置零。若不按照 6->0 的顺序赋高电平，则 flag 不会随之改变，只有前一个按键对应的 flag 已经置零时才会改变当前 flag 的值，以此控制按键顺序。

6. 蜂鸣器译码仿真



在蜂鸣器译码模块中，将时钟分为 6 个频率，作为不同的音阶，在输入的 3 位二进制数 beep 不同时，蜂鸣器的工作频率不同，对应不同的声音，beep 的值由**按键检测蜂鸣器模块**和**开关机检测模块**控制。

四、代码

```
module numchooser(clk,rst,row,col,digitscan,digitcath,Button,SW_7,beepout);

input clk,rst;
input [6:0] Button;              //按键
input SW_7;                          //总开关
output [7:0]row;                 //点阵的行总线
output [7:0]col;                 //点阵的列总线
output [7:0]digitscan;       //数码管段选总线
output [7:0]digitcath;       //数码管位选总线
output beepout;                  //蜂鸣器总线
wire [6:0] clk_seven;        //分频时钟
wire [6:0] flag;                 //按键统计
wire [6:0] Button_pulse;   //消抖按键
wire clk_500;
//开机自检信号判断
reg begin_signal;
wire begin_signal_temp;
assign begin_signal_temp=begin_signal;

always@(posedge clk or posedge rst or negedge  SW_7) begin
if(SW_7==0)begin
    begin_signal<=0;
    end
else if(rst) begin
    begin_signal<=1;
    end
else
    begin_signal<=begin_signal;
end

//结束信号判断
reg end_signal;
wire end_signal_temp;
assign end_signal_temp=end_signal;

always@(posedge clk or posedge rst or negedge  flag[0])begin
if(flag[0]==0)begin
    end_signal<=1;
    end
else
    end_signal<=0;
end
```

```
//分频器例化
time_divider u_divide(
    .clk(clk),
    .clk_seven(clk_seven),
    .clk_500(clk_500)
);

//信号发生器例化
signal_creater u_signal(
    .clk(clk),
    .rst(rst),
    .Button_pulse(Button_pulse),
    .flag(flag),
    .Button(Button),
    .clk_500(clk_500)
    );

//译码器例化
decoder u_decode(
    .clk(clk),
    .SW_7(SW_7),
    .clk_500(clk_500),
    .rst(rst),
    .Button(Button),
    .clk_seven(clk_seven),
    .begin_signal_temp(begin_signal_temp),
    .end_signal_temp(end_signal_temp),
    .beepout(beepout),
    .flag(flag),
    .digitscan(digitscan),
    .row(row),
    .digitcath(digitcath),
    .col(col)

);
endmodule

//译码器模块
module
decoder(clk,clk_500,rst,Button,SW_7,clk_seven,begin_signal_temp,end_signal_temp,
```

```
flag,beepout,col,digitscan,row,digitcath);
input clk,clk_500,rst,begin_signal_temp,end_signal_temp;
input SW_7;
input [7:0]Button;
input [6:0]clk_seven;
input [6:0]flag;
output beepout;
output [7:0]col;
output [7:0]digitscan;
output[7:0]row;
output[7:0]digitcath;

wire [7:0]col_temp1;            //列线 1
wire [7:0]col_temp2;            //列线 2
wire [7:0]digit_seg1;       //段选线 1
wire [7:0]digit_seg2;       //段选线 2
wire [7:0]row_temp1;                //行扫线 1
wire [7:0]digitcath_temp1;      //位选线 1
wire [2:0]beep_1;                       //蜂鸣器次级分支线
wire [2:0]beep1;
wire [2:0]beep2;                //蜂鸣器最低分支线
wire row_signal;
wire digit_signal;
wire [2:0]beep_2;
wire [2:0]beep;
//连线
assign col=begin_signal_temp?col_temp1:col_temp2;
assign digitscan = begin_signal_temp  ? digit_seg1  : digit_seg2;
//连线
assign row=row_signal?8'b11111111:row_temp1;
assign digitcath = digit_signal ? 8'b11111111:digitcath_temp1  ;

//连线
assign beep_1=begin_signal_temp?beep2:beep1;    //按键音
assign beep=end_signal_temp?beep_2:beep_1;

judgeFlag_beep u_beep2(                         //按键统计（蜂鸣器）
    .clk(clk),
    .Button(Button),
    .beep(beep2)
    );

start_check    u_start(                         //开机自检解码
    .clk_2Hz(clk_seven[6]),
```

```
        .SW_7(SW_7),
        .col(col_temp2),
        .digit_seg(digit_seg2),
        .beep(beep1)
        );

end_check u_end(                                    //结束解码
        .clk_2Hz(clk_seven[6]),
        .end_signal(end_signal_temp),
        .beep_2(beep_2),
        .row_signal(row_signal),
        .digit_signal(digit_signal)
        );

decode_lattice      u7(                             //点阵解码
        .clk_500(clk_500),
        .clk_2Hz(clk_seven[6]),
        .rst(rst),
        .row(row_temp1),
        .col(col_temp1),
        .flag(flag[6])
);

decode_digit u_digit(                               //数码管解码
        .clk_500(clk_500),
        .flag(flag[6:0]),
        .clk1(clk_seven[0]),
        .clk2(clk_seven[1]),
        .clk3(clk_seven[2]),
        .clk4(clk_seven[3]),
        .clk5(clk_seven[4]),
        .clk6(clk_seven[5]),
        .rst(rst),
        .digit_seg(digit_seg1),
        .digit_cath(digitcath_temp1)
);
beep_on u_beep(                                     //按键音解码
        .clk(clk),
        .clk_500(clk_500),
        .rst(rst),
        .beep(beep),
        .beepout(beepout)
);
endmodule
```

//信号发生器模块

```
module signal_creater(clk,rst,Button_pulse,flag,Button,clk_500);
input clk,rst,clk_500;
output [6:0] flag;
input     [6:0]Button;
output [6:0]Button_pulse;

judgeFlag u_flag(                                    //按键判断统计
      .clk(clk),
      .rst(rst),
      .Button_pulse(Button_pulse),
      .flag(flag)
      );
debounce #(.N(7)) u_debounce(                  //消抖
      .clk(clk),
      .rst(rst),
      .key(~Button),
      .key_pulse(Button_pulse),
      );
Endmodule
```

//分频器模块

```
module time_divider(clk,clk_seven,clk_500);
input clk;
output [6:0]clk_seven;
output clk_500;
divider3 #(.N(50000)) u_clk_500(          //500Hz
      .clkin(clk),
      .clkout(clk_500)
      );

divider2 #(.N(5)) u_clk_6(                     //2Hz
      .clkin(clk_seven[0]),
      .clkout(clk_seven[6])
      );

divider2 #(.N(1)) u_clk_5(                     //2.5Hz
      .clkin(clk_seven[3]),
      .clkout(clk_seven[5])
      );

divider2 #(.N(3)) u_clk_4(                     //3.3Hz
```

```
    .clkin(clk_seven[0]),
    .clkout(clk_seven[4])
    );

divider2 #(.N(1)) u_clk_3(                  //5Hz
    .clkin(clk_seven[2]),
    .clkout(clk_seven[3])
    );

divider2 #(.N(1)) u_clk_2(                  //10Hz
    .clkin(clk_seven[0]),
    .clkout(clk_seven[2])
    );

divider4 #(.N(20)) u_clk_1(                 //12.5Hz
    .clkin(clk_500),
    .clkout(clk_seven[1])
    );

divider1 #(.N(1250000)) u_clk_0(            //20Hz
    .clkin(clk),
    .clkout(clk_seven[0])
    );

endmodule
```

//按键音译码器

```
module beep_on(clk,clk_500,rst,beep,beepout);
input clk,clk_500,rst;
input[2:0]beep;
reg[14:0] div_buffer1;
reg[14:0] div_buffer2;
reg[14:0] div_buffer3;
reg[14:0] div_buffer4;
reg[13:0] div_buffer5;
reg[13:0] div_buffer6;
reg clk_temp1;
reg clk_temp2;
reg clk_temp3;
reg clk_temp4;
reg clk_temp5;
reg clk_temp6;
output reg beepout;
```

```
always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer1 <= 1'b0;
           clk_temp1 <= 1'b0;
       end
   else if(div_buffer1==15'd23912)
      begin
         div_buffer1 <= 1'b0;
         clk_temp1 <= ~clk_temp1;
       end
   else
      div_buffer1 <=div_buffer1+1'b1;
end

always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer2 <= 1'b0;
           clk_temp2 <= 1'b0;
       end
   else if(div_buffer2==15'd21283)
      begin
         div_buffer2 <= 1'b0;
         clk_temp2 <= ~clk_temp2;
       end
   else
      div_buffer2 <=div_buffer2+1'b1;
end
always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer3 <= 1'b0;
           clk_temp3 <= 1'b0;
       end
   else if(div_buffer3==15'd18961)
      begin
         div_buffer3 <= 1'b0;
         clk_temp3 <= ~clk_temp3;
       end
   else
```

```
        div_buffer3 <=div_buffer3+1'b1;
end
always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer4 <= 1'b0;
           clk_temp4 <= 1'b0;
       end
   else if(div_buffer4==15'd17897)
      begin
         div_buffer4<= 1'b0;
         clk_temp4<= ~clk_temp4;
       end
   else
      div_buffer4<=div_buffer4+1'b1;
end
always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer5 <= 1'b0;
           clk_temp5 <= 1'b0;
       end
   else if(div_buffer5==15'd15944)
      begin
         div_buffer5<= 1'b0;
         clk_temp5<= ~clk_temp5;
       end
   else
      div_buffer5<=div_buffer5+1'b1;
end
always@(posedge clk or posedge rst) //分频器
begin
   if(rst)
      begin
          div_buffer6 <= 1'b0;
           clk_temp6<= 1'b0;
       end
   else if(div_buffer6==15'd14205)
      begin
         div_buffer6<= 1'b0;
         clk_temp6 <= ~clk_temp6;
       end
```

```
   else
      div_buffer6<=div_buffer6+1'b1;
end

always@(posedge clk )begin
case(beep)
      3'b001:beepout<=clk_temp1;
      3'b010:beepout<=clk_temp2;
      3'b011:beepout<=clk_temp3;
      3'b100:beepout<=clk_temp4;
      3'b101:beepout<=clk_temp5;
      3'b110:beepout<=clk_temp6;
      3'b111:beepout<=clk_500;
      default:beepout<=0;
      endcase
end
endmodule
```

//开机自检译码

```
module start_check(clk_2Hz,SW_7,col,digit_seg,beep);
input clk_2Hz,SW_7;
output reg [7:0]col;
output reg[7:0] digit_seg;
output reg[2:0] beep;
reg [3:0] code;
reg flag;

always @(posedge clk_2Hz) begin
      if(SW_7==0)
            begin
                  flag<=1'b1;
                  code<=0;
                  col<=8'b0000_0000;
                  digit_seg<=8'b00000000;
                  beep<=0;
            end
      else if(SW_7==1)
      begin
                  if(code==6)
                        begin
                              flag<=0;
```

```
                col<=0;
                digit_seg<=0;
                beep<=0;
            end
        else if(flag) begin
            col<=~col;
            digit_seg <= ~digit_seg;
            beep<=beep+1;
            code <= code+1;
            end
        else
        begin
        col<=0;
        digit_seg<=0;
        beep<=0;
        end
    end
else begin
    code<=code;
    beep<=0;
end
end
endmodule
```

//结束译码

```
module end_check(clk_2Hz,end_signal,beep_2,row_signal,digit_signal);
input clk_2Hz;//要传 2Hz
input end_signal;
output reg row_signal;
output reg digit_signal;
output reg [2:0]beep_2;
reg [3:0] code;
reg temp;

always@(posedge clk_2Hz)begin
if(end_signal==0)
    begin
        temp<=1'b1;
        code<=0;
        row_signal<=0;
        digit_signal<=0;
        beep_2<=0;
    end
```

```
    else if(end_signal==1)
    begin
            if(code==6)
                begin
                    temp<=0;
                    row_signal<=0;
                    digit_signal<=0;
                    beep_2<=0;
                end
            else if(temp) begin
                row_signal<=~row_signal;
                digit_signal <= ~digit_signal;
                beep_2<=beep_2+1;
                code <= code+1;
                end
            else
            begin
            row_signal<=0;
            digit_signal<=0;
            beep_2<=0;
            end
        end
    else begin
        code<=code;
        beep_2<=0;
    end
    end
endmodule
```

//按键声音判断统计

```
module judgeFlag_beep(clk,Button,beep);
input clk;
input [6:0]Button;
output reg[2:0] beep;
always@(posedge clk) begin
    if(Button[0])begin
                beep<=3'b001;
            end
    else if(Button[1])begin
                beep<=3'b010;
            end
    else if(Button[2])begin
                beep<=3'b011;
```

```
            end
    else if(Button[3])begin
                beep<=3'b100;
            end
    else if(Button[4])begin
                beep<=3'b101;
            end
    else if(Button[5])begin
                beep<=3'b110;
            end
    else if(Button[6])begin
                beep<=3'b111;
            end
    else begin
            beep<=0;
        end
end
endmodule
```

//按键判断模块

```
module judgeFlag(clk,rst,flag,Button_pulse);
input clk,rst;
input [6:0]Button_pulse;
output reg [6:0]flag;

initial
    begin
    flag<=7'b111_1111;
    end

always @(posedge clk or posedge rst ) begin
    if (rst) begin
        // reset
        flag <= 7'b111_1111;
    end
    else if (Button_pulse[6])  begin
        flag[6]<=0;
    end
    else if (Button_pulse[5]  & flag[6] == 0) begin
        flag[5]<=0;
    end
    else if (Button_pulse[4]  & flag[5] == 0) begin
        flag[4]<=0;
```

```
    end
    else if (Button_pulse[3]  & flag[4] == 0) begin
        flag[3]<=0;
    end
    else if (Button_pulse[2]  & flag[3] == 0) begin
        flag[2]<=0;
    end
    else if (Button_pulse[1]  & flag[2] == 0) begin
        flag[1]<=0;
    end
    else if (Button_pulse[0]  & flag[1] == 0) begin
        flag[0]<=0;
    end
    else begin
        flag<=flag;
    end
end

endmodule
```

//数码管译码器

```
module          decode_digit(clk_500,flag,clk1,clk2,clk3,clk4,clk5,clk6,rst,digit_seg,
digit_cath);
    input clk_500;
    input clk1,clk2,clk3,clk4,clk5,clk6;
    input rst;
    input [5:0]flag;
    output reg[7:0] digit_seg;
    output reg[7:0] digit_cath;
    reg [23:0] code;
    reg [3:0] digit;
    reg [2:0] cnt;

always@(posedge clk_500 or posedge rst)//计数器
    begin
        if (rst)
            cnt<=0;
        else if(cnt==5)
            cnt<=0;
        else cnt<=cnt+1;
    end

always@(posedge clk1 or posedge rst )
```

```
    begin
        if(rst )
        code[3:0]<=4'b0;
        else if(flag[0]==1)
            begin
                if(code[3:0]==4'h9)
                    begin
                        code[3:0]<=0;
                    end
                else
                    code[3:0] <= code[3:0]+1;
            end
        else
            code[3:0]<=code[3:0];
    end

always@(posedge clk2 or posedge rst )
    begin
        if(rst )
        code[7:4]<=0;
        else if(flag[1]==1)
            begin
                if(code[7:4]==4'h9)
                    begin
                        code[7:4]<=0;
                    end
                else
                    code[7:4] <= code[7:4]+1;
            end
        else
            code[7:4]<=code[7:4];
    end

always@(posedge clk3 or posedge rst )
    begin
        if(rst )
        code[11:8]<=0;
        else if(flag[2]==1)
            begin
                if(code[11:8]==4'h9)
                    begin
                        code[11:8]<=0;
                    end
                else
```

```
                code[11:8] <= code[11:8]+1;
            end
        else
            code[11:8]<=code[11:8];
    end

always@(posedge clk4 or posedge rst )
    begin
        if(rst )
        code[15:12]<=0;
        else if(flag[3]==1)
            begin
                if(code[15:12]==4'h9)
                    begin
                        code[15:12]<=0;
                    end
                else
                    code[15:12] <= code[15:12]+1;
            end
        else
            code[15:12]<=code[15:12];
    end

always@(posedge clk5 or posedge rst )
    begin
        if(rst )
        code[19:16]<=0;
        else if(flag[4]==1)
            begin
                if(code[19:16]==4'h9)
                    begin
                        code[19:16]<=0;
                    end
                else
                    code[19:16] <= code[19:16]+1;
            end
        else
            code[19:16]<=code[19:16];
    end

always@(posedge clk6 or posedge rst )
    begin
        if(rst )
        code[23:20]<=4'ha;
```

```
        else if(flag[5]==1)
            begin
                if(code[23:20]==4'hf)
                    begin
                        code[23:20]<=4'ha;
                    end
                else
                    code[23:20] <= code[23:20]+1;
            end
        else
            code[23:20]<=code[23:20];
    end
always @(clk_500) begin
    case (digit)
    4'h0:    digit_seg <= 8'b11111100;//显示 0
     4'h1:    digit_seg <= 8'b01100000;//显示 1
     4'h2:    digit_seg <= 8'b11011010;//显示 2
     4'h3:    digit_seg <= 8'b11110010;//显示 3
     4'h4:    digit_seg <= 8'b01100110;//显示 4
     4'h5:    digit_seg <= 8'b10110110;//显示 5
     4'h6:    digit_seg <= 8'b10111110;//显示 6
     4'h7:    digit_seg <= 8'b11100000;//显示 7
     4'h8:    digit_seg <= 8'b11111110;//显示 8
     4'h9:    digit_seg <= 8'b11110110;//显示 9
     4'hA:    digit_seg <= 8'b11101110;//显示 A
     4'hB:    digit_seg <= 8'b10011100;//显示 C
     4'hC:    digit_seg <= 8'b10011110;//显示 E
     4'hD:    digit_seg <= 8'b10001110;//显示 F
     4'hE:    digit_seg <= 8'b01101110;//显示 H
     4'hF:    digit_seg <= 8'b00011100;//显示 L
    default:;
    endcase
end

always @(posedge clk_500) begin
    case(cnt)
    3'h0: begin
        digit_cath<=8'b1111_1110;
        digit<=code[3:0];
    end
    3'h1: begin
        digit_cath<=8'b1111_1101;
        digit<=code[7:4];
    end
```

```
        3'h2: begin
            digit_cath<=8'b1111_1011;
            digit<=code[11:8];
        end
        3'h3: begin
            digit_cath<=8'b1111_0111;
            digit<=code[15:12];
        end
        3'h4: begin
            digit_cath<=8'b1110_1111;
            digit<=code[19:16];
        end
        3'h5: begin
            digit_cath<=8'b1101_1111;
            digit<=code[23:20];
        end
        default:;
        endcase
end

        endmodule
```

//点阵显示译码器

```
module decode_lattice(clk_500,clk_2Hz, rst, row, col,flag);
        input clk_500;
        input clk_2Hz;
        input rst,flag;
        reg [3:0] code;
        output reg [7:0] row;
        output reg [7:0] col;
        reg [63:0] col_temp;
        reg [2:0] cnt;

        always@(posedge clk_2Hz or posedge rst )       //计数器 1
            begin
                if(rst)
                    begin
                        code<=0;
                    end
                else if(flag==1)
                    begin
                        if(code==4'h4)
                            begin
```

```
                    code<=0;
                end
            else begin
                code <= code+1;
                end
        end
    else begin
        code<=code;
    end
end

always @(posedge clk_500) begin          //计数器 2
if (rst) begin
cnt<=0;
end
else begin
cnt<=cnt+1;
end
end

always @(posedge clk_500) begin          //扫描
    case(cnt)
        3'h0: begin
            row<=8'b1111_1110;
            col<=col_temp[7:0];
        end
        3'h1: begin
            row<=8'b1111_1101;
            col<=col_temp[15:8];
        end
        3'h2: begin
            row<=8'b1111_1011;
            col<=col_temp[23:16];
        end
        3'h3: begin
            row<=8'b1111_0111;
            col<=col_temp[31:24];
        end
        3'h4: begin
            row<=8'b1110_1111;
            col<=col_temp[39:32];
        end
        3'h5: begin
            row<=8'b1101_1111;
```

```
            col<=col_temp[47:40];
        end
        3'h6: begin
            row<=8'b1011_1111;
            col<=col_temp[55:48];
        end
        3'h7: begin
            row<=8'b0111_1111;
            col<=col_temp[63:56];
        end
        default:;
    endcase
    end

always @(clk_500) begin          //换字
    case(code)
        4'h0:                    col_temp                    <=
64'b00010000_11111111_01111110_01000010_01111110_01010100_11010010_0011
0000;
        4'h1:                    col_temp                    <=
64'b11000100_01011111_10010001_00011111_01010001_01010000_10100000_101
00000;
        4'h2:                    col_temp                    <=
64'b00011000_11111111_00011000_01111110_00000000_01111110_01000010_0111
1110;
        4'h3:                    col_temp                    <=
64'b00100100_11111111_00010000_11111100_01010110_10100101_01100100_110
01100;
        4'h4:                    col_temp                    <=
64'b01001001_01001001_01001001_01001001_01001001_01001001_01001001_10
001001;
    endcase
end
endmodule
```

//按键消抖模块

```
module debounce (clk,rst,key,key_pulse);
parameter N    =    1;                    //要消除的按键的数量
input clk;
input rst;
input [N-1:0]     key;                    //输入的按键
output [N-1:0]     key_pulse;             //按键动作产生的脉冲
reg [N-1:0]     key_rst_pre;              //定义一个寄存器型变量存储上一个
```

```
触发时的按键值
reg [N-1:0]     key_rst;                          //定义一个寄存器变量储存储当前
时刻触发的按键值
wire [N-1:0]     key_edge;                        //检测到按键由高到低变化是产
生一个高脉冲

//利用非阻塞赋值特点，将两个时钟触发时按键状态存储在两个寄存器变量中
always @(posedge clk    or    posedge rst)
  begin
       if (rst) begin
            key_rst <= {N{1'b1}};                //初始化时给 key_rst 赋值全为
1，{}中表示 N 个 1
            key_rst_pre <= {N{1'b1}};
       end
       else begin
            key_rst  <= key;                     //第一个时钟上升沿触发之后
key 的值赋给 key_rst,同时 key_rst 的值赋给 key_rst_pre
            key_rst_pre <= key_rst;              //非阻塞赋值。相当于经过两个时
钟触发，key_rst 存储的是当前时刻 key 的值，key_rst_pre 存储的是前一个时钟的
key 的值
       end
   end
assign    key_edge = key_rst_pre & (~key_rst);//脉冲边沿检测。当 key 检测到下降
沿时，key_edge 产生一个时钟周期的高电平
reg [17:0]cnt;                                   //产生延时所用的计数器，时钟 500Hz，
要延时 20ms 左右时间，至少需要 4 位计数器
//此处改动 17->3
//产生 20ms 延时，当检测到 key_edge 有效是计数器清零开始计数
always @(posedge clk or posedge rst)
   begin
       if(rst)
            cnt <= 18'h0;                        //此处改动 18->4
       else if(key_edge)
            cnt <= 18'h0;
       else
            cnt <= cnt + 1'h1;
       end

reg        [N-1:0]     key_sec_pre;              //延时后检测电平寄存器变量
reg        [N-1:0]     key_sec;


//延时后检测 key，如果按键状态变低产生一个时钟的高脉冲。如果按键状态是
高的话说明按键无效
```

```
always @(posedge clk    or    posedge rst)
 begin
      if (rst)
            key_sec <= {N{1'b1}};
      else if (cnt==18'h3ffff)                //此处改动 18‘h3ffff->4'hf
            key_sec <= key;
 end
always @(posedge clk    or    posedge rst)
 begin
      if (rst)
            key_sec_pre <= {N{1'b1}};
      else
            key_sec_pre <= key_sec;
end
assign    key_pulse = key_sec_pre & (~key_sec);

endmodule
```

//分频器模块 1

```
module divider1(clkin, clkout);
parameter N = 1;
input clkin;
output reg clkout;
reg [21:0] cnt;
initial
begin
clkout=0;
cnt=0;
end
always @(posedge clkin) begin
     if (cnt==N) begin
          clkout <= !clkout;
          cnt <= 0;
     end
     else begin
          cnt <= cnt + 1;
     end
end
endmodule
```

//分频器模块 2

```
module divider2(clkin, clkout);
parameter N = 1;
input clkin;
output reg clkout;
reg [2:0] cnt;
initial
begin
clkout=0;
cnt=0;
end
always @(posedge clkin) begin
    if (cnt==N) begin
        clkout <= !clkout;
        cnt <= 0;
    end
    else begin
        cnt <= cnt + 1;
    end
end
endmodule
```

//分频器模块 3

```
module divider3(clkin, clkout);
parameter N = 1;
input clkin;
output reg clkout;
reg [15:0] cnt;
initial
begin
clkout=0;
cnt=0;
end
always @(posedge clkin) begin
    if (cnt==N) begin
        clkout <= !clkout;
        cnt <= 0;
    end
    else begin
        cnt <= cnt + 1;
    end
end
endmodule
```

//分频器模块 4

```
module divider4(clkin, clkout);
parameter N = 1;
input clkin;
output reg clkout;
reg [4:0] cnt;
initial
begin
clkout=0;
cnt=0;
end
always @(posedge clkin) begin
    if (cnt==N) begin
        clkout <= !clkout;
        cnt <= 0;
    end
    else begin
        cnt <= cnt + 1;
    end
end
endmodule
```

五、功能说明及资源利用情况

（一）实现功能：

基本要求：

1、 用 SW7 作为选号机开关，打开开关 SW7 后选号机自检：8*8 点阵和数码管 DISP7~DISP0 全亮 0.5S 熄灭 0.5S 重复三次，进入待机状态；

2、 使用按键 BTN7 进入选号状态，按以下顺序进行选号，当前面的号码未选定时，后 面的按键无效。具体要求如下：

a) 8*8 点阵轮流显示“京”“ 沪”“ 吉”“ 苏”“ 川”五个汉字，每个汉字显示停留 时间 0.5S，按动 BTN6 选中当前显示的汉字，该汉字稳定显示；

b) 数码管 DISP5 上轮流显示“A”“ C”“ E”“ F”“ H”“ L”六个大写字母，每个字 母显示停留时间 0.4S，按动 BTN5 选中当前显示的字母，该字母稳定显示；

c) 数码管 DISP4 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.3S，按 动 BTN4 选中当前显示的数字，该数字稳定显示；

d) 数码管 DISP3 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.2S，按 动 BTN3 选中当前显示的数字，该数字稳定显示；

e) 数码管 DISP2 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.1S，按 动 BTN2 选中当前显示的数字，该数字稳定显示；

f) 数码管 DISP1 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.08S，按 动 BTN1 选中当前显示的数字，该数字稳定显示；

g) 数码管 DISP0 上轮流显示“0~9”十个数字，每个数字显示停留时间 0.05S，按 动 BTN0 选中当前显示的数字，该数字稳定显示；

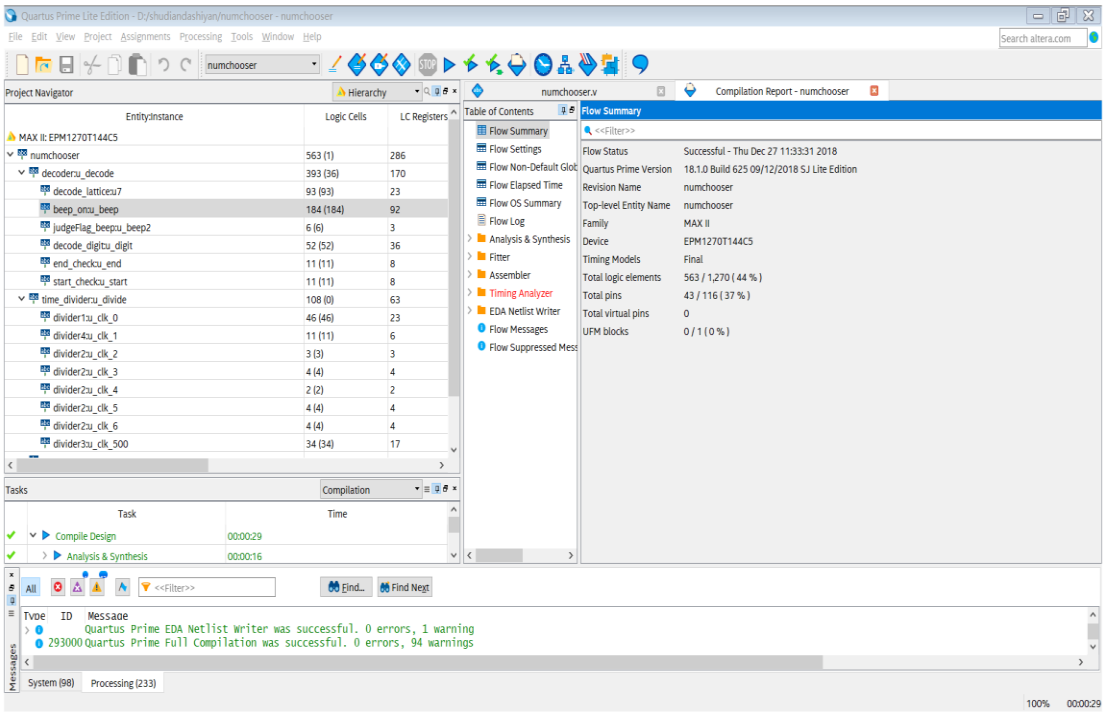
3、 DISP0 内容选定后表示所有内容选择完毕，所有内容整体以 2Hz 闪烁三次以示提醒， 然后稳定显示；

4、 使用按键 BTN7 可以重新进入选号状态，再一次进行选号。

提高要求：

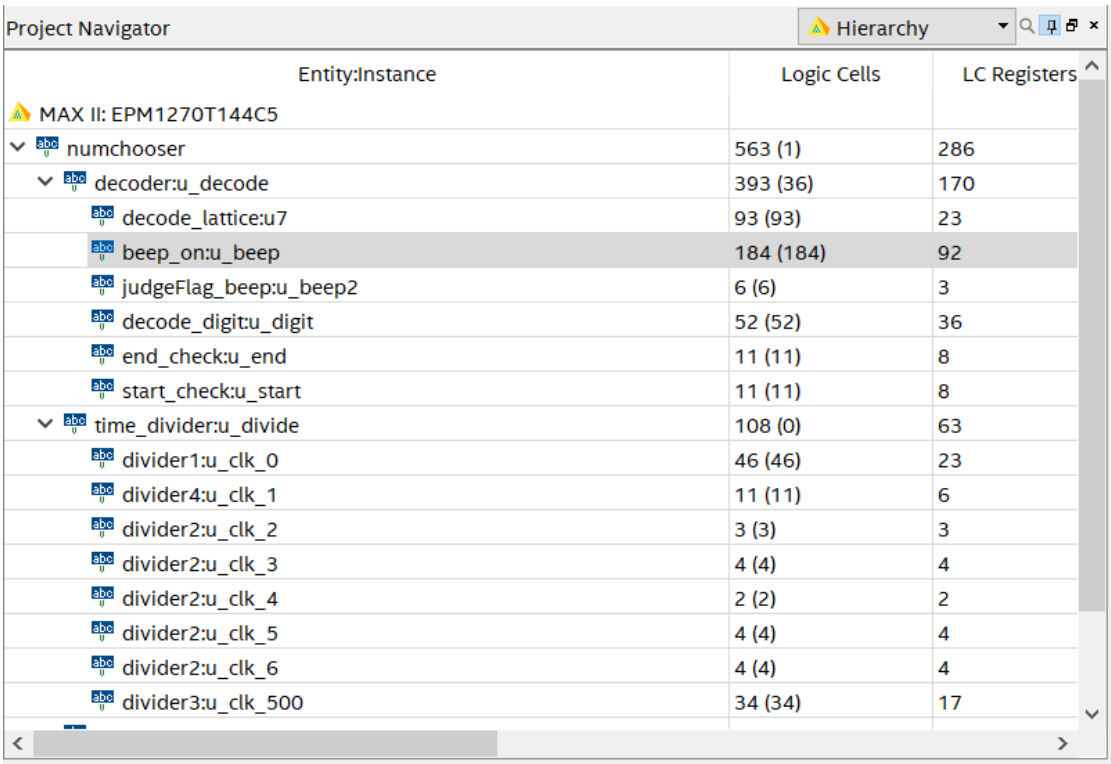
自检过程、各项内容滚动时、内容选定后进行闪烁提醒时伴有适当的音乐，各个按 键按下时伴有按键音。

（二）资源利用情况



资源占用率：44%（563/1270）

引脚占用率：37%（43/116）



管脚分配

Node Name	Direction	Location	I/O Bank	Fitter Locatior	I/O Standard	Reserved	urrent Streng	ict Preservati
Button[6]	Input	PIN_123	2	PIN_123	3.3-V LVTTL		16mA ...ault)	
Button[5]	Input	PIN_122	2	PIN_122	3.3-V LVTTL		16mA ...ault)	
Button[4]	Input	PIN_121	2	PIN_121	3.3-V LVTTL		16mA ...ault)	
Button[3]	Input	PIN_91	3	PIN_91	3.3-V LVTTL		16mA ...ault)	
Button[2]	Input	PIN_89	3	PIN_89	3.3-V LVTTL		16mA ...ault)	
Button[1]	Input	PIN_20	1	PIN_20	3.3-V LVTTL		16mA ...ault)	
Button[0]	Input	PIN_61	4	PIN_61	3.3-V LVTTL		16mA ...ault)	
SW_7	Input	PIN_125	2	PIN_125	3.3-V LVTTL		16mA ...ault)	
beepout	Output	PIN_60	4	PIN_60	3.3-V LVTTL		16mA ...ault)	
clk	Input	PIN_18	1	PIN_18	3.3-V LVTTL		16mA ...ault)	
col[7]	Output	PIN_22	1	PIN_22	3.3-V LVTTL		16mA ...ault)	
col[6]	Output	PIN_21	1	PIN_21	3.3-V LVTTL		16mA ...ault)	
col[5]	Output	PIN_16	1	PIN_16	3.3-V LVTTL		16mA ...ault)	
col[4]	Output	PIN_15	1	PIN_15	3.3-V LVTTL		16mA ...ault)	
col[3]	Output	PIN_14	1	PIN_14	3.3-V LVTTL		16mA ...ault)	
col[2]	Output	PIN_13	1	PIN_13	3.3-V LVTTL		16mA ...ault)	
col[1]	Output	PIN_12	1	PIN_12	3.3-V LVTTL		16mA ...ault)	
col[0]	Output	PIN_11	1	PIN_11	3.3-V LVTTL		16mA ...ault)	
digitcath[7]	Output	PIN_31	1	PIN_31	3.3-V LVTTL		16mA ...ault)	
digitcath[6]	Output	PIN_30	1	PIN_30	3.3-V LVTTL		16mA ...ault)	
digitcath[5]	Output	PIN_70	4	PIN_70	3.3-V LVTTL		16mA ...ault)	
digitcath[4]	Output	PIN_69	4	PIN_69	3.3-V LVTTL		16mA ...ault)	
digitcath[3]	Output	PIN_68	4	PIN_68	3.3-V LVTTL		16mA ...ault)	
digitcath[2]	Output	PIN_67	4	PIN_67	3.3-V LVTTL		16mA ...ault)	
digitcath[1]	Output	PIN_66	4	PIN_66	3.3-V LVTTL		16mA ...ault)	
digitcath[0]	Output	PIN_63	4	PIN_63	3.3-V LVTTL		16mA ...ault)	
digitscan[7]	Output	PIN_62	4	PIN_62	3.3-V LVTTL		16mA ...ault)	
digitscan[6]	Output	PIN_59	4	PIN_59	3.3-V LVTTL		16mA ...ault)	
digitscan[5]	Output	PIN_58	4	PIN_58	3.3-V LVTTL		16mA ...ault)	
digitscan[4]	Output	PIN_57	4	PIN_57	3.3-V LVTTL		16mA ...ault)	
digitscan[3]	Output	PIN_55	4	PIN_55	3.3-V LVTTL		16mA ...ault)	
digitscan[2]	Output	PIN_53	4	PIN_53	3.3-V LVTTL		16mA ...ault)	
digitscan[1]	Output	PIN_52	4	PIN_52	3.3-V LVTTL		16mA ...ault)	
digitscan[0]	Output	PIN_51	4	PIN_51	3.3-V LVTTL		16mA ...ault)	
row[7]	Output	PIN_1	1	PIN_1	3.3-V LVTTL		16mA ...ault)	
row[6]	Output	PIN_2	1	PIN_2	3.3-V LVTTL		16mA ...ault)	
row[5]	Output	PIN_3	1	PIN_3	3.3-V LVTTL		16mA ...ault)	
row[4]	Output	PIN_4	1	PIN_4	3.3-V LVTTL		16mA ...ault)	
row[3]	Output	PIN_5	1	PIN_5	3.3-V LVTTL		16mA ...ault)	
row[2]	Output	PIN_6	1	PIN_6	3.3-V LVTTL		16mA ...ault)	
row[1]	Output	PIN_7	1	PIN_7	3.3-V LVTTL		16mA ...ault)	
row[0]	Output	PIN_8	1	PIN_8	3.3-V LVTTL		16mA ...ault)	
rst	Input	PIN_124	2	PIN_124	3.3-V LVTTL		16mA ...ault)	
<<new node>>								

六、故障及问题分析

（一）双线串行分频的延迟问题

在分 12.5Hz 和 20Hz 之后的频率时,12.5Hz 的原频率来自 500Hz，20Hz 的原频率来自 50MHz，这样采用双线串行分频会导致时间信号之间由一定的延迟，对信号造成一定影响，但由于延迟很小，分频倍数很高，延迟可以忽略，在仿真和实验中也没有出现延迟问题，于是决定就采用双线串行分频。

（二）开机自检和结束的选择问题

在设计后期，对顶层模块进行优化的时候，关于开机自检的方式产生了疑问，开机时制造一个开始信号，却不清楚要怎么传，后来想到了单刀双掷开关的方法，利用开始信号做位开关，至于信号则来自于两个不同的译码器。而且在开机自检时只需要控制列和段选，行和位选仍然交给正常工作的译码器完成，在关机时只需要控制行扫描和位选,其他仍交给正常工作的译码器完成。这样两个译码器虽是并行，同一时刻只有一个译码器的编码可以输出，从而解决了开机自检和结束的选择问题。

（三）数码管译码器不同位不同频的问题

为了达到数码管译码器不同位不同频的效果，需要设计 6 个不同的计数器,在不同的位用不同的计数器来计数,从而实现不同频变化，同频扫描。

（四）仿真问题

在仿真过程中，频率的倍数问题成了最困难的地方，由于计算机

计算 1×10^7数量级的计算量较大，导致在用 modelsim 仿真过程中需要将原来项目中的分频倍数进行调整，由于我使用串行分频，需要改动的地方不多，但是仿真图象一直有一些小问题，后来经过细致的检查和排除，终于明白不仅仅是我的分频器中的分频倍数要改，在所有通过系统时钟做计数器的地方都需要将系统时钟换为频率更低的时钟，经过修改最终仿真成功。

七、总结和讨论

本次数电实验完成选号机的各项功能总体来说完成较好，虽然过程很曲折但是在曲折的过程中能学到更多的知识。

此次实验我基本掌握了 Verilog 这门硬件开发语言，对我后续的项目制作很有帮助。同时，我还在实验中懂得了信号仿真的优势：仿真对于实验帮助是巨大的，但我由于后期才开始做仿真，所以做后才真正运用仿真和 RTL 图去修改我的代码。在仿真过程中既可以找到系统的 BUG，又可以找到不完整或不完美的地方去修改，因此在以后的数电或其他专业课学习过程中我会提前进行仿真的学习。

在制作选号机的过程中，我对于分频器，按键消抖，信号发生器，译码器和状态机等数电学习中本来模糊的概念有了更深的理解，同时在实验中能运用到数电课内学过的只是也很满足，一个功能一个功能的完善最能给实验者以成就感，最终完成所有功能再去优化的过程既富有挑战又充满创意，只有这样才可以做出一个好的项目。