北京郵電大學

《数字系统设计实验》实验报告

题目：基于 CPLD 的模拟风暖式浴霸控制器的设计与实现

班　　级 _____

姓　　名 _____

学　　号 _____

指导老师 _____

2021 年 12 月

目录

第一章 实验要求

1.1 实验目的

利用 CPLD 器件和实验开发板，设计并实现一个用于浴室的风暖式浴霸控制器。

1.2 任务要求

基本要求：

1、 SW6 作为控制器的整机开关，SW6=0 为关机状态，打开 SW6（SW6=1）后，点阵全红、全绿交替全亮以 2Hz 闪烁，8 个数码管同时亮“8”以 2Hz 频率闪烁，同时 16 个发光二极管 LD15-LD0 也以 2Hz 频率闪烁，2 秒后进入待机状态：点阵、数码管和发光二极管 LD15-LD0 全灭；

2、 待机状态（SW6=1）下，风暖式浴霸的基本功能有照明、换气、风暖、强暖和干燥等 5 种功能，其中换气、风暖、强暖和干燥这 4 个功能互斥，照明功能与其它 4 个功能不互斥，即在换气、风暖、强暖或干燥这 4 个功能下同时可以有照明功能。

3、 在 SW6=1 时，用按键开关 BTN7 控制照明灯 LD6 的亮灭，即 BTN7 为照明开关，LD6 为照明灯。在 SW6=0 时，照明灯 LD6 不能被点亮，始终处于熄灭状态。

4、 待机状态（SW6=1）下，用不同的点阵动画模拟浴霸的 4 个功能：换气、风暖、强暖和干燥功能的点阵动画过程分别如图 1、2、3 和 4 所示，每种点阵动画的一个循环的完成时间为 2 秒，即动画中每一帧的显示时间是 0.5 秒。

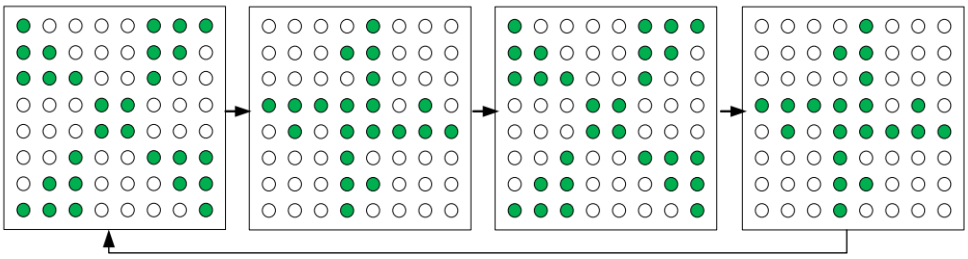


图 1　换气功能的点阵动画过程

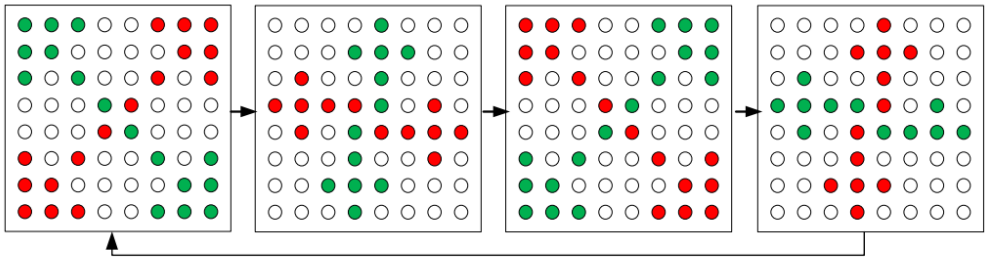


图 2　风暖功能的点阵动画过程

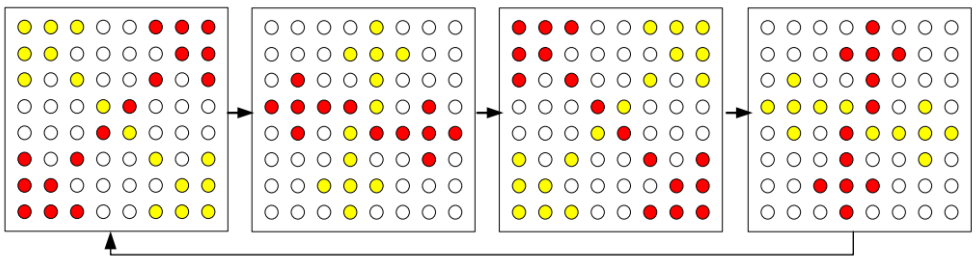


图 3　强暖功能的点阵动画过程

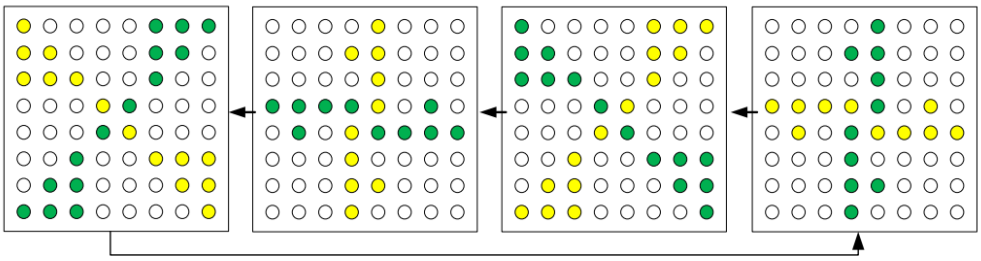


图 4　干燥功能的点阵动画过程

5、 在待机状态（SW6=1）下，用按键开关 BTN6、BTN5、BTN4 和 BTN3 分别实现换气、风暖、强暖和干燥等 4 个功能的启动、停止；在换气功能状态下按一下 BTN6 或在干燥功能状态下按一下 BTN3，点阵动画立即停止；在风暖功能状态下，按一下 BTN5，点阵动画延时 2 秒后停止；在强暖功能状态下，按一下 BTN4，点阵动画延时 4 秒后停止；延时时间以倒计时的方式显示在数码管 DISP5 上。

6、 待机状态（SW6=1）下，按键开关 BTN6、BTN5、BTN4 和 BTN3 同时也是换气、风暖、强暖和干燥这 4 个功能之间的切换按键：以换气功能为例，在换气状态时，若按动按键 BTN5、BTN4 或 BTN3，则浴霸功能分别切换为风暖、强暖或干燥这 3 个功能之一，按动按键 BTN6 则浴霸停止换气功能，进入待机状态；其它功能之间的切换依此类推。

7、 系统工作流程合理，工作稳定。

提高要求：

1、 给按键加上不同的按键音；

2、 增加音效，模拟浴霸风扇的噪音；

3、 自拟其他功能。

模块电路要求：

在 8×8 双色点阵上设计并实现图 1-4 其中一种动画，要求完成仿真并在实验板上下载显示。

第二章 系统设计

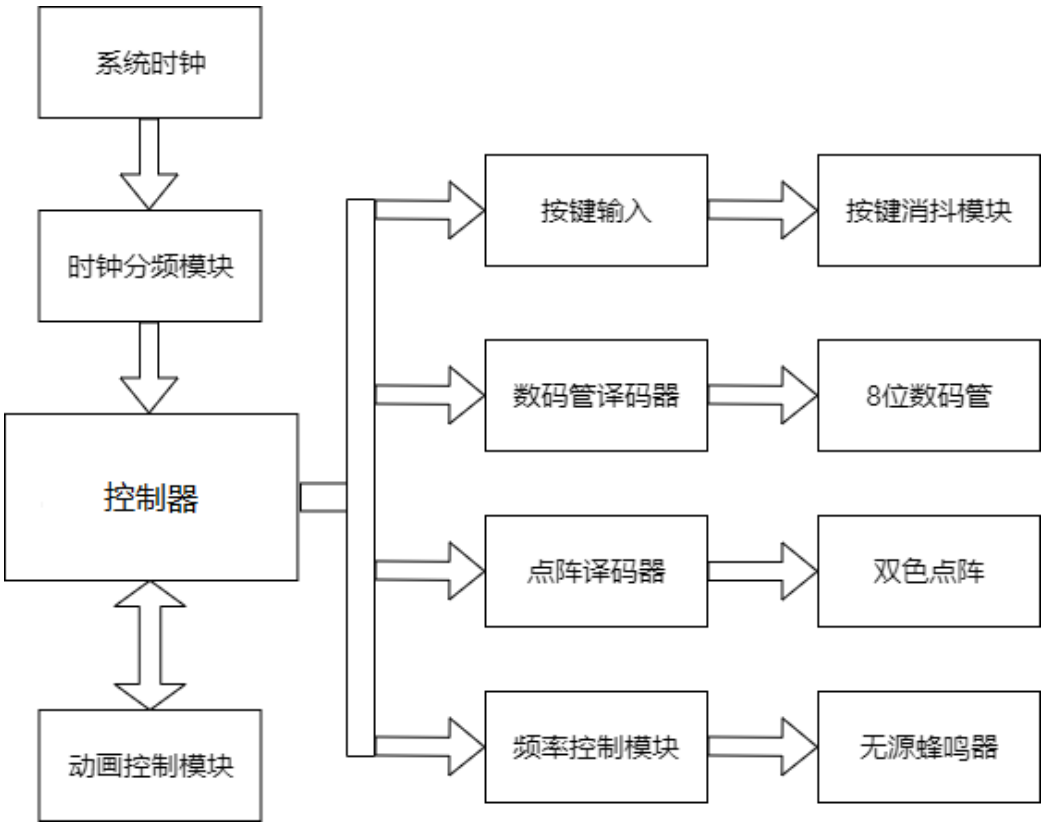
2.1 设计思路

实验总体设计思路采用自顶向下设计法，首先根据实验要求确定初步方案，构建系统框图；其次进行系统划分，并对子系统功能进行描述；最后进行逻辑描述，完成具体设计。本次实验旨在利用按键切换模式并显示相应的点阵动画，点击与当前模式相同的按键会切换为待机模式，并且在强暖和风暖模式切换至待机时会先延时一定时间。除此之外，提高要求为实现浴霸风速的噪音以及按键的效果音，利用 4*4 键盘替代按钮。

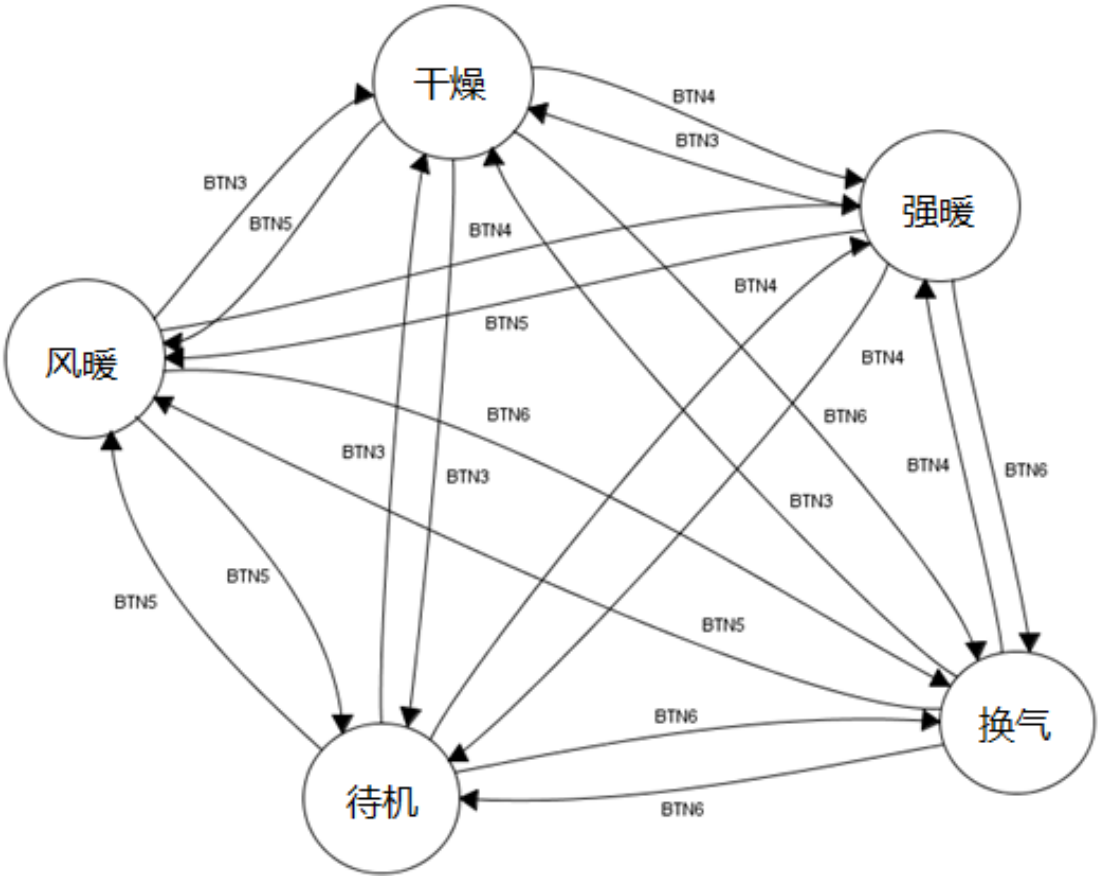
综合以上，确定系统总体输入为系统时钟、SW6 开关、BTN7 至 BTN0 八个开关、4*4 键盘的输入行；总体输出为 4*4 键盘的扫描列、LED 灯 LD15 至 LD0 共十六位、双色点阵绿色以及红色输出行、双色点阵选通列、八段数码管共阴角电平、八位数码管选通信号、无源蜂鸣器电平。

2.2 系统设计

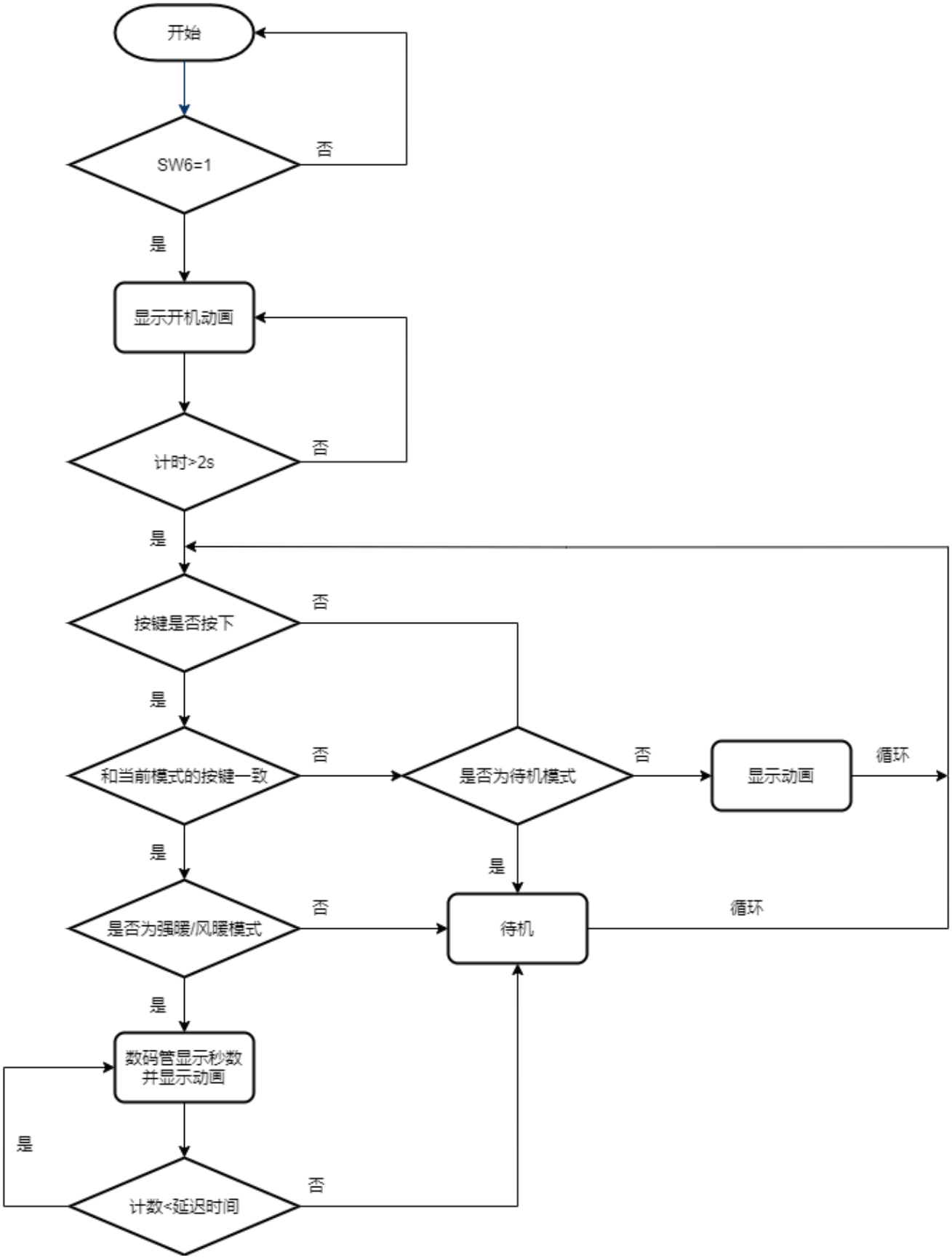
2.2.1 总体框图



2.2.2 状态转移图

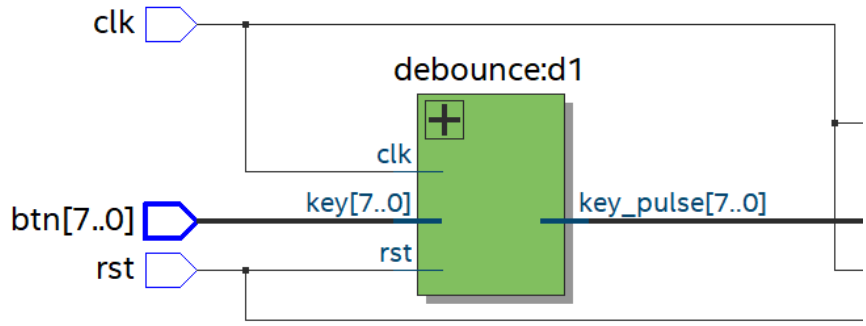


2.2.3 系统流程图


开始
SW6=1
否
是
显示开机动画
计时>2s
否
是
按键是否按下
否
是
和当前模式的按键一致
否
是否为待机模式
否
显示动画
循环
是
是否为强暖/风暖模式
否
待机
循环
是
数码管显示秒数
并显示动画
是
计数<延迟时间
否

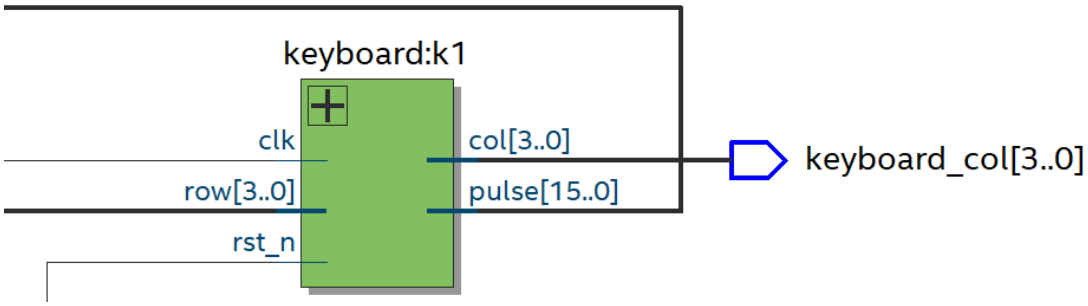
2.3 分块设计

2.3.1 消抖模块(debounce)



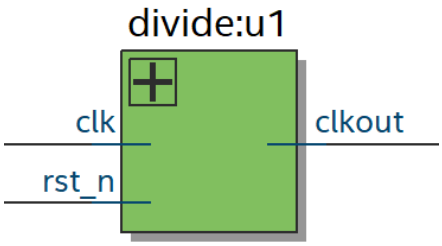
消抖模块输入为系统时钟、复位信号、所需消抖的按键脉冲；输出为消抖后的按键脉冲。消抖的原理为脉冲边沿检测后进行 20ms 延时，利用延时前和延时后电平产生按键脉冲。

2.3.2 4*4 键盘模块(keyboard)



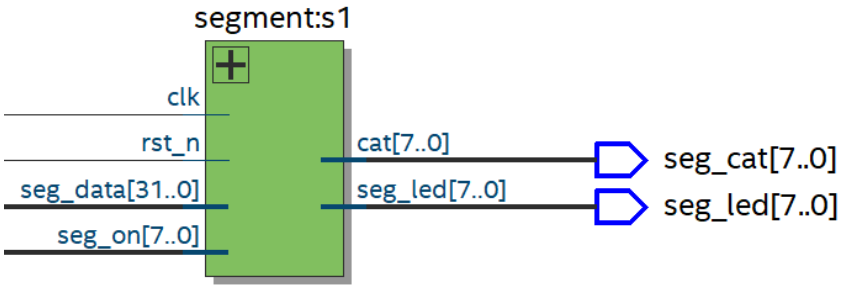
4*4 键盘模块的输入为系统时钟、复位信号、键盘行信号；输出为键盘选通列信号、16 位消抖后的键盘脉冲。其原理是构造状态机循环读取每一行是否有按键按下，若有，则锁存此刻的行扫描值与列读取值，再通过译码器将数值转换为对应位置的按键信号。4*4 键盘一样有按键抖动的问题，所以需对键盘是否有按键按下的信号进行消抖，最终输出消抖后的 16 位键盘脉冲。

2.3.3 分频器(division)



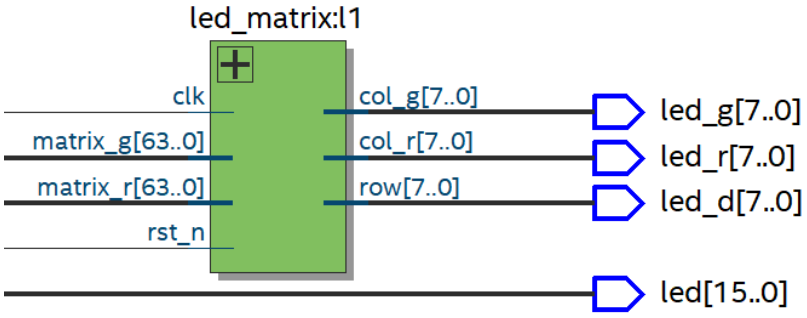
分频器的输入为时钟、复位信号；输出信号为分频后的时钟脉冲，其原理为利用计数器计数产生延迟后的脉冲。参数 N 为分频系数，输入时钟频率除以 N 等于输出时钟频率。

2.3.4 数码管译码器(segment)



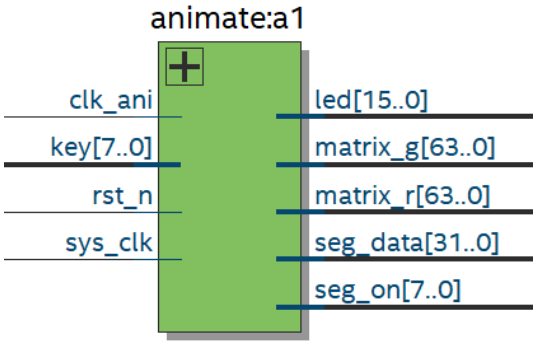
数码管译码器输入为扫描时钟、复位信号、8 位数码管所需显示的数值、8 位数码管的使能信号；输出信号为数码管的选通信号、八段数码管的电平。选通信号用于选择数码管，当扫描时钟频率高于 50Hz 后人眼无法分辨。

2.3.5 双色点阵译码器(led_matrix)



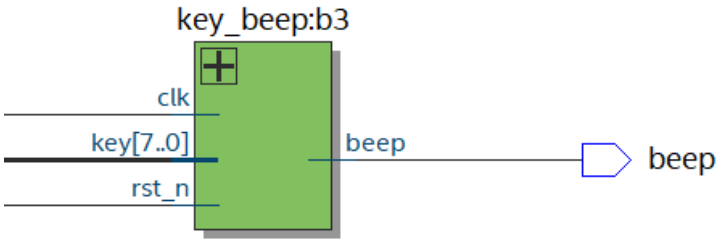
双色点阵译码器的输入为扫描时钟、8*8 红色 led 的电平、8*8 绿色 led 的电平以及复位信号；输出为双色点阵的 8 位红色、绿色 led 电平值以及点阵扫描行信号。通过扫描行信号选通每一行的 led，当扫描频率大于 50Hz 时人眼无法辨别。

2.3.6 动画控制模块(animate)



动画控制模块的输入为系统时钟、动画时钟、复位信号以及按键脉冲；输出为 led 的电平、8 位数码管每一位所需显示的数值、8 位数码管的使能信号以及传递给双色点阵模块的 8*8 点阵灯信号。

2.3.7 蜂鸣器模块(key_beep)



蜂鸣器模块输入为系统时钟、复位信号以及按键脉冲；输出为蜂鸣器电平。蜂鸣器模块内部将输入的按键脉冲转换为相应的蜂鸣器输出频率，再利用计数器对系统时钟脉冲延时输出得到所需要的蜂鸣器电平。其中蜂鸣器若需要输出音乐，需根据下图音阶频率对照输出。

音阶频率表（单位赫兹 hz）

	1	2	3	4	5	6	7	8	9
A	27.5	55	110	220	440	880	1760	3520	7040
#A	29.135	58.27	116.541	233.082	466.164	932.328	1864.655	3729.31	7458.62
B	30.868	61.735	123.471	246.942	493.883	987.767	1975.533	3951.066	7902.132
C	32.703	65.406	130.813	261.626	523.251	1046.502	2093.004	4186.009	
#C	34.648	69.296	138.591	277.183	554.365	1108.731	2217.461	4434.922	
D	36.708	73.416	146.832	293.665	587.33	1174.659	2349.318	4698.636	
#D	38.891	77.782	155.563	311.127	622.254	1244.598	2489.016	4978.032	
E	41.203	82.407	164.814	329.629	659.255	1318.52	2637.02	5274.04	
F	43.654	87.307	174.614	349.228	698.456	1396.913	2793.826	5587.652	
#F	46.249	92.499	184.997	369.994	739.989	1479.978	2959.955	5919.91	
G	48.999	97.999	195.998	391.995	783.991	1567.982	3135.437	6270.874	
#G	51.913	103.826	207.652	415.305	830.609	1661.219	3322.437	6644.874	

注：绿色部分为钢琴 88 键常用音阶。

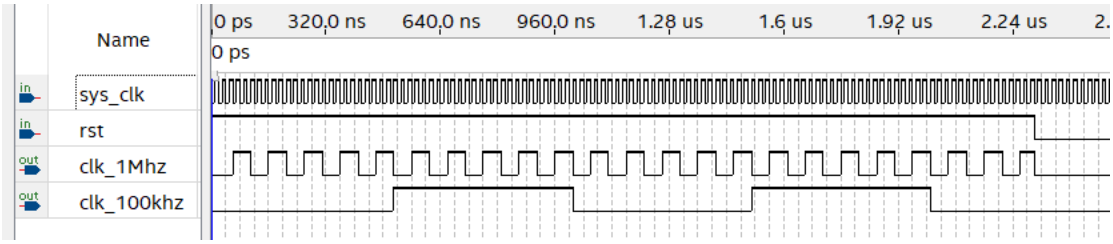
第三章 仿真波形及波形分析

3.1 仿真参数调整

由于仿真时间最多为 100us，若直接使用原程序进行仿真将无法看到对应的功能波形，故需将原程序的分频时钟频率进行适当修改，本次仿真中将动画时钟由 2Hz 修改为 200kHz，其余分频时钟均修改为 10MHz，即分频系数修改为 250 和 5 且仿真中的 10us 对应为原程序的 1s。同时，仿真输入能够产生完美的按键电平脉冲，且不需要消抖，所以在仿真时可以删除消抖模块直接使用按键输入。

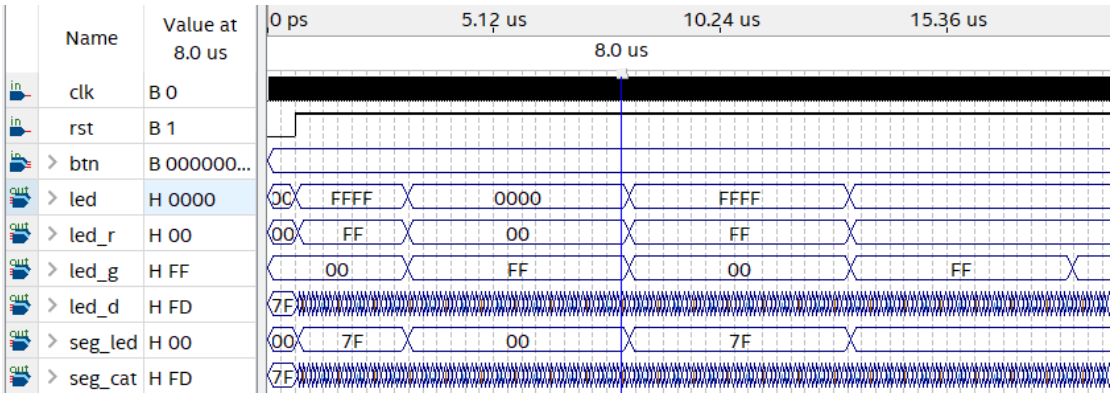
3.2 仿真波形及波形分析

3.2.1 分频器



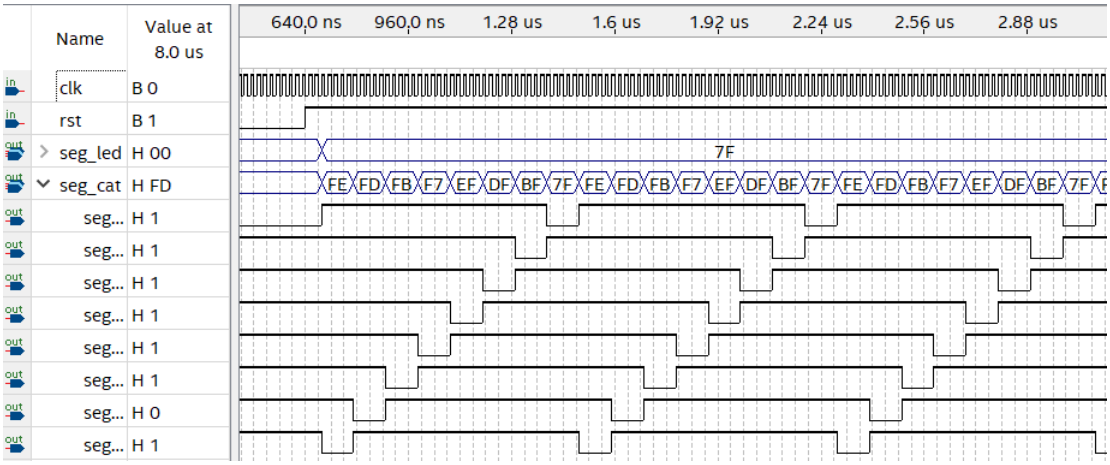
可见当系统时钟频率为 50MHz、分频系数分别为 50、5 时，分频器产生频率为 1MHz、100kHz 的脉冲信号，且低有效的复位信号能够成功复位分频器。

3.2.2 开机动画



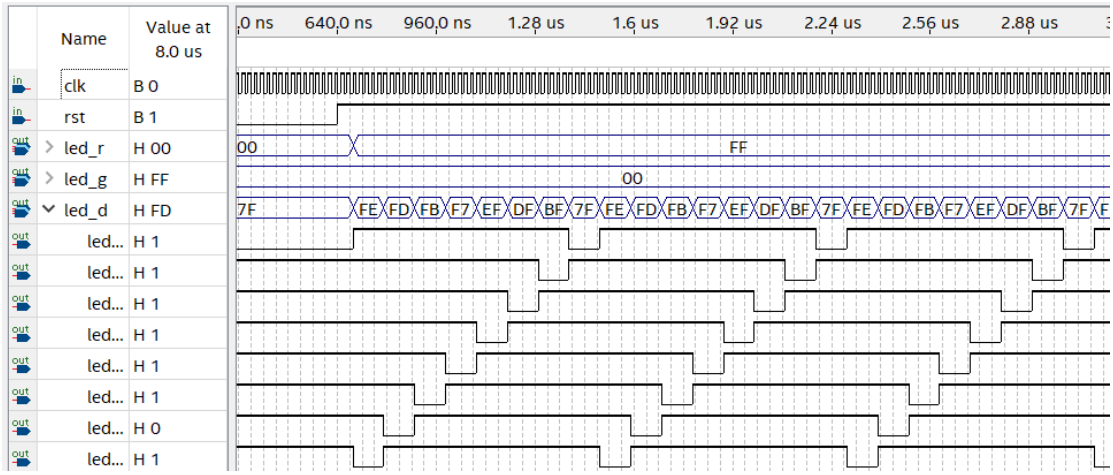
可见 8 位数码管、16 位 LED、双色点阵在复位信号无效后均以 200kHz 闪烁（即周期为 5us），延时 20us 秒，与实际情况对应的正是 2Hz，延时 2 秒，符合实验要求。

3.2.3 数码管选通扫描



可见数码管选通信号以 10MHz 频率扫描，其结果与实际频率 100kHz 相对应，符合实验要求。

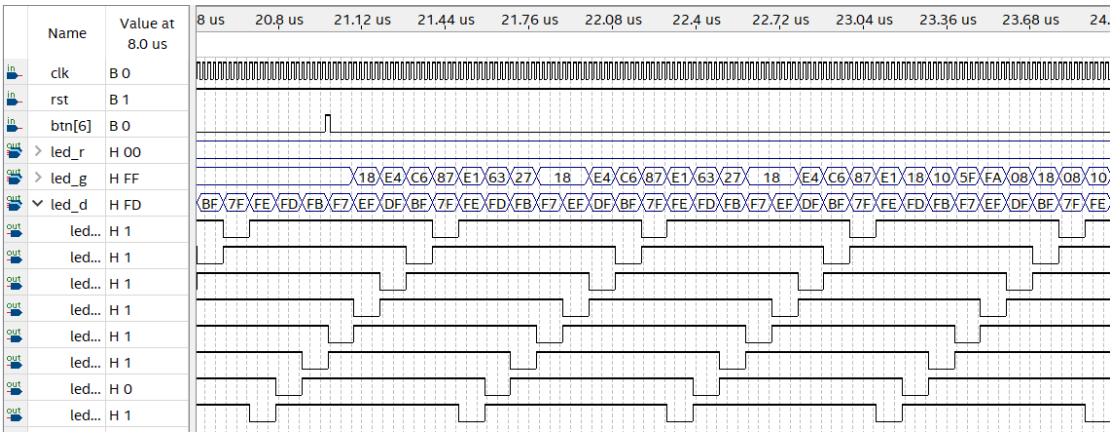
3.2.4 双色点阵列扫描



可见双色点阵列输出信号以 10MHz 频率扫描，其结果与实际频率 100kHz 相对应，符合

实验要求。

3.2.5 换气功能点阵动画



可见点阵红色行没有输出、点阵绿色行循环输出十六进制 87C6_E418_1827_63E1 和 0818_08FA_5F10_1810，且延时 5us 后切换动画，其对应的正是换气功能点阵动画的显示情况，下面其他三种功能同理可得，之后不再阐述。

注：

十六进制 87C6_E418_1827_63E1 的二进制为：

10000111 11000110 11100100 00011000 00011000 00100111 01100011 11100001

十六进制 0818_08FA_5F10_1810 的二进制为：

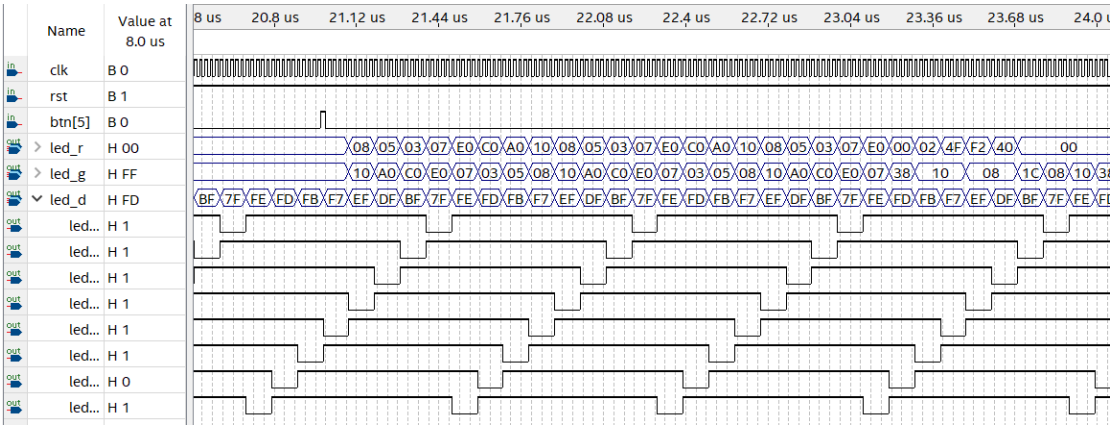
00001000 00011000 00001000 11111010 01011111 00010000 00011000 00010000

经排列后可见：

```
1 0 0 0 0 1 1 1     0 0 0 0 1 0 0 0
1 1 0 0 0 1 1 0     0 0 0 1 1 0 0 0
1 1 1 0 0 1 0 0     0 0 0 0 1 0 0 0
0 0 0 1 1 0 0 0     1 1 1 1 1 0 1 0
0 0 0 1 1 0 0 0     0 1 0 1 1 1 1 1
0 0 1 0 0 1 1 1     0 0 0 1 0 0 0 0
0 1 1 0 0 0 1 1     0 0 0 1 1 0 0 0
1 1 1 0 0 0 0 1     0 0 0 1 0 0 0 0
```

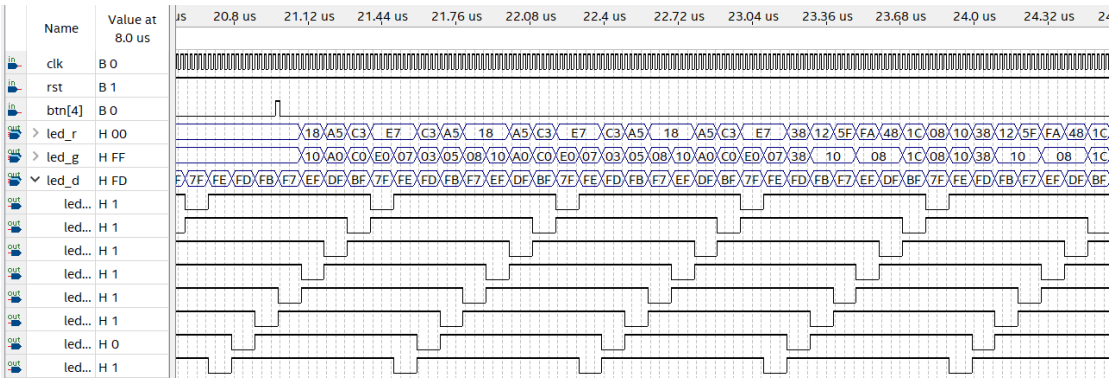
与实验题目要求相符

3.2.6 风暖功能点阵动画



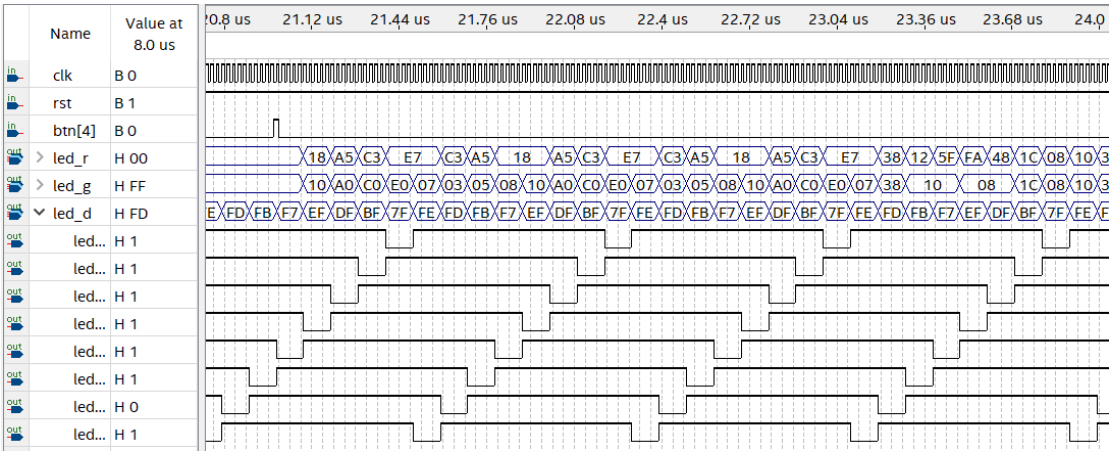
可见点阵红色和绿色行输出对应的正是风暖功能点阵动画的显示情况，结果与实验题目要求相符。

3.2.7 强暖功能点阵动画



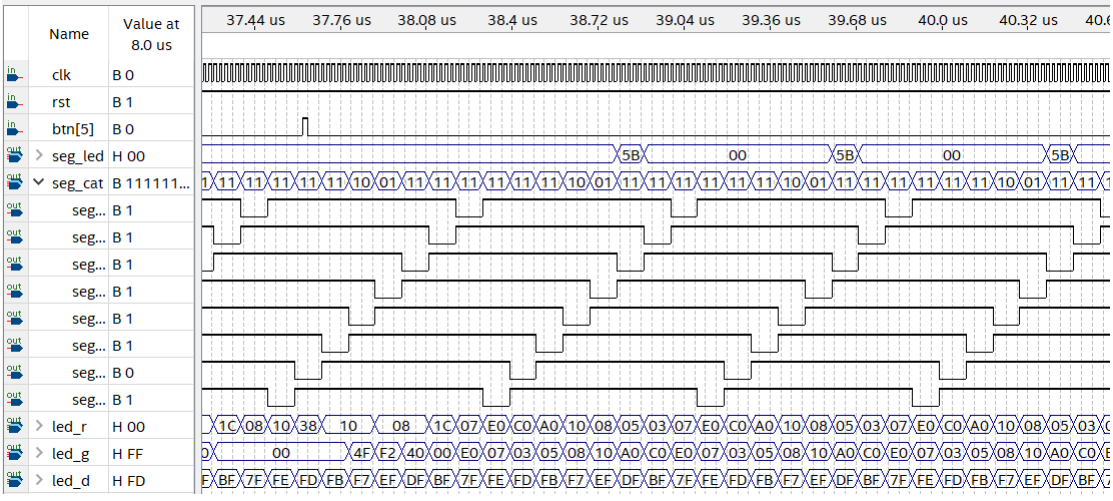
可见点阵红色和绿色行输出对应的正是强暖功能点阵动画的显示情况，结果与实验题目要求相符。

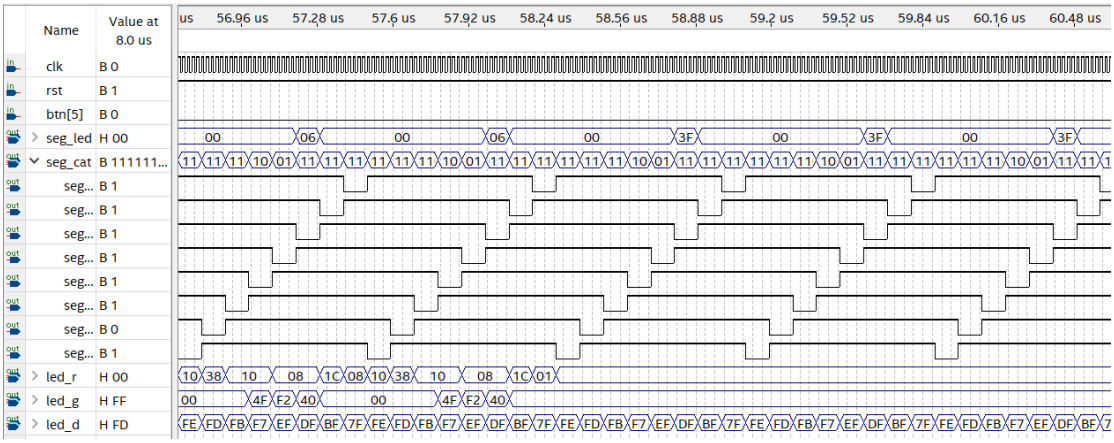
3.2.8 干燥功能点阵动画



可见点阵红色和绿色行输出对应的正是干燥功能点阵动画的显示情况，结果与实验题目要求相符。

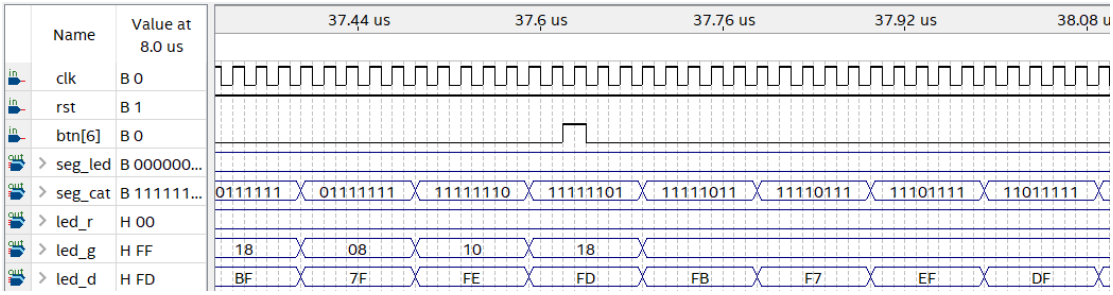
3.2.9 数码管延时秒数显示





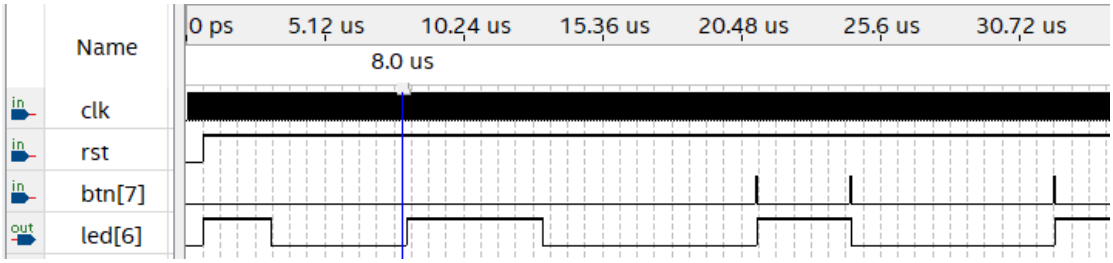
可见当在强暖模式下按下按钮 BTN5，数码管 DISP5 输出十六进制 5B、06、3F，其对应的正是数码管显示的数字 2,1,0，且当秒数为零时点阵输出关闭。这与实际强暖模式关闭时状态一致，结果与实验题目要求相符。

3.2.10 点阵立即停止动画



可见在换气模式下按下 BTN6，点阵立即停止输出，数码管依旧没有输出，这对应的正是实际情况下的点阵动画停止，结果与实验题目要求相符。

3.2.11 切换照明灯



可见当 BTN7 按下时照明灯 LD6 切换开关，20us 前的跳变为系统的开启动画闪烁。

第四章 代码

4.1 消抖处理(debounce.v)

```
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  negedge 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;
          end
    end
    assign key_edge = (~key_rst_pre) & (key_rst);
    //脉冲边沿检测。


    reg[19:0] cnt;//产生延时所用的计数器,系统时钟 50MHz,要延时 20ms 左右时间,至少需要 20 位计数器
    //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 negedge rst)begin
        if(!rst)cnt <= 20'h0;
        else if(key_edge)cnt <= 20'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  negedge rst)begin
        if(!rst)key_sec <= {N{1'b1}};
        else if(cnt==20'hfffff)key_sec <= key;
    end
    always @(posedge clk  or  negedge 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
```

4.2 4*4 键盘(keyboard.v)

```
module keyboard(
  input          sys_clk,        // 系统时钟
  input            rst_n,
  input      [3:0] row,                // 矩阵键盘 行
  output reg [3:0] col,                // 矩阵键盘 列
  output [15:0] pulse               // 键盘消抖后的脉冲
);

//++++++++++++++++++++++++++++++++++++++
// 分频部分 开始
//++++++++++++++++++++++++++++++++++++++
reg [19:0] cnt;                         // 计数子
always @ (posedge sys_clk, negedge rst_n)
  if (!rst_n)cnt <= 0;
  else cnt <= cnt + 1'b1;
```

```
wire key_clk = cnt[18];               // (2^20/50M = 21)ms
//-------------------------------------
// 分频部分 结束
//-------------------------------------


//+++++++++++++++++++++++++++++++++++++
// 状态机部分 开始
//+++++++++++++++++++++++++++++++++++++
// 状态数较少，独热码编码
localparam NO_KEY_PRESSED = 6'b000_001;  // 没有按键按下
localparam SCAN_COL0      = 6'b000_010;  // 扫描第 0 列
localparam SCAN_COL1      = 6'b000_100;  // 扫描第 1 列
localparam SCAN_COL2      = 6'b001_000;  // 扫描第 2 列
localparam SCAN_COL3      = 6'b010_000;  // 扫描第 3 列
localparam KEY_PRESSED    = 6'b100_000;  // 有按键按下

reg [5:0] current_state, next_state;    // 现态、次态

always @ (posedge key_clk, negedge rst_n)
  if (!rst_n)current_state <= NO_KEY_PRESSED;
  else current_state <= next_state;

// 根据条件转移状态
always @ *
  case (current_state)
    NO_KEY_PRESSED :                    // 没有按键按下
        if (row != 4'hF)next_state = SCAN_COL0;
        else next_state = NO_KEY_PRESSED;
    SCAN_COL0 :                         // 扫描第 0 列
        if (row != 4'hF)next_state = KEY_PRESSED;
        else next_state = SCAN_COL1;
    SCAN_COL1 :                         // 扫描第 1 列
        if (row != 4'hF)next_state = KEY_PRESSED;
        else next_state = SCAN_COL2;
    SCAN_COL2 :                         // 扫描第 2 列
        if (row != 4'hF)next_state = KEY_PRESSED;
        else next_state = SCAN_COL3;
    SCAN_COL3 :                         // 扫描第 3 列
        if (row != 4'hF)next_state = KEY_PRESSED;
        else next_state = NO_KEY_PRESSED;
    KEY_PRESSED :                       // 有按键按下
        if (row != 4'hF)next_state = KEY_PRESSED;
        else next_state = NO_KEY_PRESSED;
  endcase

reg [3:0] keyboard_val;            // 按键值
reg       key_pressed_flag;             // 键盘按下标志
reg [3:0] col_val, row_val;             // 列值、行值

// 根据次态，给相应寄存器赋值
always @ (posedge key_clk, negedge rst_n)
  if (!rst_n)
  begin
    col             <= 4'h0;
    key_pressed_flag <=    0;
  end
  else
```

```
    case (next_state)
      NO_KEY_PRESSED :                  // 没有按键按下
      begin
        col              <= 4'h0;
        key_pressed_flag <=    0;       // 清键盘按下标志
      end
      SCAN_COL0 : col <= 4'b1110;       // 扫描第 0 列
      SCAN_COL1 : col <= 4'b1101;       // 扫描第 1 列
      SCAN_COL2 : col <= 4'b1011;       // 扫描第 2 列
      SCAN_COL3 : col <= 4'b0111;       // 扫描第 3 列
      KEY_PRESSED :                     // 有按键按下
      begin
        col_val          <= col;        // 锁存列值
        row_val          <= row;        // 锁存行值
        key_pressed_flag <= 1;          // 置键盘按下标志
      end
    endcase
//-------------------------------------
// 状态机部分 结束
//-------------------------------------


//+++++++++++++++++++++++++++++++++++++
// 扫描行列值部分 开始
//+++++++++++++++++++++++++++++++++++++
always @ (posedge key_clk, negedge rst_n)
  if (!rst_n)
    keyboard_val <= 4'h0;
  else
    if(key_pressed_flag)begin
      case ({col_val, row_val})
        8'b1110_1110 : keyboard_val <= 4'hC;
        8'b1110_1101 : keyboard_val <= 4'h8;
        8'b1110_1011 : keyboard_val <= 4'h4;
        8'b1110_0111 : keyboard_val <= 4'h0;

        8'b1101_1110 : keyboard_val <= 4'hD;
        8'b1101_1101 : keyboard_val <= 4'h9;
        8'b1101_1011 : keyboard_val <= 4'h5;
        8'b1101_0111 : keyboard_val <= 4'h1;

        8'b1011_1110 : keyboard_val <= 4'hE;
        8'b1011_1101 : keyboard_val <= 4'hA;
        8'b1011_1011 : keyboard_val <= 4'h6;
        8'b1011_0111 : keyboard_val <= 4'h2;

        8'b0111_1110 : keyboard_val <= 4'hF;
        8'b0111_1101 : keyboard_val <= 4'hB;
        8'b0111_1011 : keyboard_val <= 4'h7;
        8'b0111_0111 : keyboard_val <= 4'h3;
      endcase
  end
//-------------------------------------
//  扫描行列值部分 结束
//-------------------------------------

wire key_pulse;
//实例化按键消抖
```

```
debounce #(1) d2(
.clk(sys_clk),
.rst(rst_n),
.key(key_pressed_flag),
.key_pulse(key_pulse)
);

assign pulse =     {keyboard_val==4'hF&key_pulse,
              keyboard_val==4'hE&key_pulse,
              keyboard_val==4'hD&key_pulse,
                keyboard_val==4'hC&key_pulse,
                keyboard_val==4'hB&key_pulse,
                keyboard_val==4'hA&key_pulse,
                keyboard_val==4'h9&key_pulse,
                keyboard_val==4'h8&key_pulse,
                keyboard_val==4'h7&key_pulse,
                keyboard_val==4'h6&key_pulse,
                keyboard_val==4'h5&key_pulse,
                keyboard_val==4'h4&key_pulse,
                keyboard_val==4'h3&key_pulse,
                keyboard_val==4'h2&key_pulse,
                keyboard_val==4'h1&key_pulse,
                keyboard_val==4'h0&key_pulse};
//assign pulse = keyboard_val | {16{key_pulse}};
endmodule
```

4.3 分频器(division.v)

```
module divide(clk,rst_n,clkout);
   input clk,rst_n;                          //输入信号，频率为 50MHz
   output clkout;                            //输出信号，可以连接到 LED 观察分频的时钟
    reg[WIDTH-1:0] cnt_p,cnt_n;        //cnt_p 为上升沿触发时的计数器，cnt_n 为下降沿触发时的计数器
    reg clk_p,clk_n;                   //clk_p 为上升沿触发时分频时钟，clk_n 为下降沿触发时分频时钟
    parameter WIDTH = 3;                        //计数器的位数，计数的最大值为 2**WIDTH-1
    parameter N = 5;                           //分频系数，请确保 N < 2**WIDTH-1，否则计数会溢出

    //上升沿触发时计数器的控制
    always @ (posedge clk or negedge rst_n)begin
        if(~rst_n)cnt_p<=0;
        else if(cnt_p==(N-1))cnt_p<=0;
        else cnt_p<=cnt_p+1;         //计数器一直计数，当计数到 N-1 的时候清零，这是一个模 N 的计数器
    end

    //上升沿触发的分频时钟输出,如果 N 为奇数得到的时钟占空比不是 50%；如果 N 为偶数得到的时钟占空比为 50%
    always @ (posedge clk or negedge rst_n)begin
        if(~rst_n)clk_p<=0;
        else if(cnt_p<(N>>1))clk_p<=0;          //N>>1 表示右移一位，相当于除以 2 去掉余数
        else clk_p<=1;                             //得到的分频时钟正周期比负周期多一个 clk 时钟
    end

   //下降沿触发时计数器的控制
    always @ (negedge clk or negedge rst_n)begin
        if(~rst_n)cnt_n<=0;
        else if(cnt_n==(N-1))cnt_n<=0;
        else cnt_n<=cnt_n+1;
    end

   //下降沿触发的分频时钟输出，和 clk_p 相差半个时钟
```

```
    always @ (negedge clk)begin
        if(~rst_n)clk_n<=0;
        else if(cnt_n<(N>>1))clk_n<=0;
        else clk_n<=1;                          //得到的分频时钟正周期比负周期多一个 clk 时钟
    end

    //当 N=1 时，直接输出 clk
    //当 N 为偶数也就是 N 的最低位为 0，N（0）=0，输出 clk_p
    //当 N 为奇数也就是 N 最低位为 1，N（0）=1，输出 clk_p&clk_n。正周期多所以是相与
    assign clkout = (N==1)?clk:(N[0])?(clk_p&clk_n):clk_p;
endmodule
```

4.4 数码管译码器(segment.v)

```
module segment(
    input clk,
    input rst_n,
    input[31:0] seg_data,    //每一位的数码管数据
    input[7:0] seg_on,       //每一位的数码管开关
    output reg[7:0] seg_led,
    output reg[7:0] cat      //选通信号
);

    reg [3:0] cnt;              // 计数
    reg [7:0] seg [15:0];
    initial begin
        cat <= 8'b0111_1111;
        cnt <= 3'b000;
        seg[0] = 8'h3f;
        seg[1] = 8'h06;       //7 段显示数字  1
        seg[2] = 8'h5b;       //7 段显示数字  2
        seg[3] = 8'h4f;       //7 段显示数字  3
        seg[4] = 8'h66;       //7 段显示数字  4
        seg[5] = 8'h6d;       //7 段显示数字  5
        seg[6] = 8'h7d;       //7 段显示数字  6
        seg[7] = 8'h07;       //7 段显示数字  7
        seg[8] = 8'h7f;       //7 段显示数字  8
        seg[9] = 8'h6f;       //7 段显示数字  9
        seg[10] = 8'h77;         //7 段显示数字  A
        seg[11] = 8'h7c;         //7 段显示数字  b
        seg[12] = 8'h39;         //7 段显示数字  C
        seg[13] = 8'h5e;         //7 段显示数字  d
        seg[14] = 8'h79;         //7 段显示数字  E
        seg[15] = 8'h71;         //7 段显示数字  F
    end

    always @(posedge clk or negedge rst_n)begin
        if(~rst_n)begin
            cat <= 8'b0111_1111;
            cnt <= 3'b000;
            seg_led <= 8'h00;
        end else begin
            cat <= {cat[6:0],cat[7]};
            seg_led <= seg_on[cnt] ? seg[seg_data[(cnt+1)*4-1-:4]] : 8'h00;
            cnt <= cnt+1;
        end
    end
endmodule
```

4.5 双色点阵译码器(led_matrix.v)

```
module led_matrix(
    input clk,
    input rst_n,
    input[63:0] matrix_r,   //输入矩阵,红色
    input[63:0] matrix_g,   //输入矩阵,绿色
    output reg[7:0] col_r,  // 列,红色
    output reg[7:0] col_g,  // 列,绿色
    output reg[7:0] row         // 行
);

    reg [3:0] cnt;              // 计数
    initial begin
        row <= 8'b0111_1111;
        col_r <= 8'h00;
        col_g <= 8'h00;
        cnt <= 3'b000;
    end

    always @(posedge clk or negedge rst_n)begin
        if(~rst_n)begin
            row <= 8'b0111_1111;
            col_r <= 8'h00;
            col_g <= 8'h00;
            cnt <= 3'b000;
        end else begin
            row <= {row[6:0],row[7]};
            col_r <= matrix_r[(cnt+1)*8-1-:8];
            col_g <= matrix_g[(cnt+1)*8-1-:8];
            cnt <= cnt+1;
        end
    end
endmodule
```

4.6 点阵灯动画控制(animate.v)

```
module animate(
    input sys_clk,
    input clk_ani,
    input rst_n,
    input[7:0] key,
    output reg[63:0] matrix_r,
    output reg[63:0] matrix_g,
    output reg[31:0] seg_data,
    output reg[7:0] seg_on,
    output reg[15:0] led
);

    localparam AIR        = 3'd0 ;  //换气模式
    localparam AIR_HEAT = 3'd1 ; //风暖模式
    localparam HEAT     = 3'd2 ; //强暖模式
    localparam DRY      = 3'd3 ;    //干燥模式
    localparam STANDBY  = 3'd4 ;    //待机模式

 /*-------------------------- 按键控制开始 --------------------------*/
```

```
    reg light = 1'b0;                   //照明灯
    reg[2:0] mode = STANDBY;          //当前模式
    reg[2:0] new_mode = STANDBY;      //按键切换的新模式

    //按键对状态机的切换
    always @(posedge sys_clk or negedge rst_n)begin
        if(~rst_n) new_mode <= STANDBY;
        else begin
            case({key[6:0]})
                7'b1000000: new_mode <= (mode!=AIR? AIR  : STANDBY);
                7'b0100000:new_mode <=(mode!=AIR_HEAT?AIR_HEAT:STANDBY);
                7'b0010000: new_mode <= (mode!=HEAT ? HEAT:STANDBY);
                7'b0001000: new_mode <= (mode!=DRY? DRY: STANDBY);
                default:    new_mode <= new_mode;
            endcase
        end
    end

    //照明灯切换
    always @(posedge sys_clk or negedge rst_n)begin
        if(~rst_n)light = 1'b0;
        else if(key[7])light <= ~light;
    end
/*--------------------------- 按键控制结束 ---------------------------*/

/*--------------------------- 动画控制开始 ---------------------------*/
    reg[1:0] cnt_ani = 2'b00;       //切换动画的计数
    reg[2:0] cnt_rst = 3'b100;  //复位或启动时的计数
    reg[3:0] cnt_cls = 4'b0000;     //关闭模式的计数
    reg[63:0] ani_r[0:3][0:3];
    reg[63:0] ani_g[0:3][0:3];

    initial begin
        ani_r[0][0] = 64'h0000_0000_0000_0000;
     //二进制从左到右，需要把 col 设置引脚时，col[0]对应 col7
        ani_r[1][0] = 64'h0000_0000_0000_0000;
        ani_r[2][0] = 64'h0000_0000_0000_0000;
        ani_r[3][0] = 64'h0000_0000_0000_0000;
        ani_g[0][0] = 64'h87C6_E418_1827_63E1;
        ani_g[1][0] = 64'h0818_08FA_5F10_1810;
        ani_g[2][0] = 64'h87C6_E418_1827_63E1;
        ani_g[3][0] = 64'h0818_08FA_5F10_1810;

        ani_r[0][1] = 64'h0703_0508_10A0_C0E0;
        ani_r[1][1] = 64'h0000_40F2_4F02_0000;
        ani_r[2][1] = 64'hE0C0_A010_0805_0307;
        ani_r[3][1] = 64'h081C_0808_1010_3810;
        ani_g[0][1] = 64'hE0C0_A010_0805_0307;
        ani_g[1][1] = 64'h081C_0808_1010_3810;
        ani_g[2][1] = 64'h0703_0508_10A0_C0E0;
        ani_g[3][1] = 64'h0000_00F2_4F02_0000;

        ani_r[0][2] = 64'hE7C3_A518_18A5_C3E7;
        ani_r[1][2] = 64'h081C_48FA_5F12_3810;
        ani_r[2][2] = 64'hE7C3_A518_18A5_C3E7;
        ani_r[3][2] = 64'h081C_48FA_5F12_3810;
        ani_g[0][2] = 64'hE0C0_A010_0805_0307;
        ani_g[1][2] = 64'h081C_0808_1010_3810;
```

```
    ani_g[2][2] = 64'h0703_0508_10A0_C0E0;
    ani_g[3][2] = 64'h0000_40F2_4F02_0000;

    ani_r[0][3] = 64'h80C0_E010_0807_0301;
    ani_r[1][3] = 64'h0818_0808_1010_1810;
    ani_r[2][3] = 64'h0706_0408_1020_60E0;
    ani_r[3][3] = 64'h0000_00F2_4F00_0000;
    ani_g[0][3] = 64'h87C6_E418_1827_63E1;
    ani_g[1][3] = 64'h0818_08FA_5F10_1810;
    ani_g[2][3] = 64'h87C6_E418_1827_63E1;
    ani_g[3][3] = 64'h0818_08FA_5F10_1810;
end

//状态机 1
always @(posedge clk_ani or negedge rst_n)begin
    if(~rst_n)begin
        cnt_ani <= 2'b00;
        cnt_rst <= 3'b100;
    end else begin
        if(new_mode == STANDBY & mode != STANDBY)begin  //延时秒数
            case(mode)
           AIR:        cnt_cls <= 4'b0000;
           AIR_HEAT:cnt_cls <= cnt_cls==4'b0000 ? 4'b0101 : cnt_cls-1; //2Hz 2 秒
                HEAT:   cnt_cls <= cnt_cls==4'b0000 ? 4'b1001 : cnt_cls-1; //2Hz 4 秒
                DRY:        cnt_cls <= 4'b0000;
            endcase
        end else cnt_cls <= cnt_cls!=4'b0000 ? cnt_cls-1 : 4'b0000;
        cnt_rst <= cnt_rst != 3'b000 ? cnt_rst-1 : 3'b000;
        cnt_ani <= mode != STANDBY ? cnt_ani+1 : 2'b00;
    end
end

//状态机 2
always @(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)mode <= STANDBY;
    else if(new_mode == STANDBY & mode != STANDBY)begin  //延时秒数
        mode <= (mode==AIR||mode==DRY)||cnt_cls==4'b0001 ? new_mode : mode; //延时切换模式
    end else mode <= new_mode;
end

//输出
always @(posedge sys_clk or negedge rst_n)begin
    if(~rst_n)begin //复位
        matrix_r <= 64'h0000_0000_0000_0000;
        matrix_g <= 64'h0000_0000_0000_0000;
        seg_data <= 32'h0000_0000;
        seg_on <= 8'b0000_0000;
        led <= 16'h0000;
    end else if(cnt_rst != 3'b000)begin //启动动画
        matrix_r <= cnt_rst%2==0 ? 64'hFFFF_FFFF_FFFF_FFFF : 64'h0000_0000_0000_0000;
        matrix_g <= cnt_rst%2==1 ? 64'hFFFF_FFFF_FFFF_FFFF : 64'h0000_0000_0000_0000;
        seg_data <= 32'h8888_8888;
        seg_on <= cnt_rst%2==0 ? 8'b1111_1111 : 8'b0000_0000;   //数码管开关
        led <= cnt_rst%2==0 ? 16'hFFFF : 16'h0000;
    end else begin //动画显示
        matrix_r <= mode != STANDBY ? ani_r[cnt_ani][mode] : 64'h0000_0000_0000_0000;
        matrix_g <= mode != STANDBY ? ani_g[cnt_ani][mode] : 64'h0000_0000_0000_0000;
        seg_data <= {8'h00,1'b0,cnt_cls>>1,20'h0000};          //关闭模式显示倒计时时间
```

```
            seg_on <= cnt_cls == 4'b0000 ? 8'b0000_0000 : 8'b0010_0000;
            led <= {9'h00,light,6'h00};                                  //保持照明灯
        end
    end
 /*-------------------------- 动画控制结束 --------------------------*/
endmodule
```

4.7 按键控制蜂鸣器(key_beep.v)

```
module key_beep(
    input sys_clk,
    input rst_n,
    input [7:0]key,
    output reg beep
);

localparam BTN1     = 95556;    //按键音 1  523.251Hz C5
localparam BTN2   = 85131;  //按键音 2  587.33Hz D5
localparam BTN3     = 75843;    //按键音 3  659.255Hz E5
localparam BTN4     = 71586;    //按键音 4  698.456HZ F5
localparam BTN5     = 63776;    //按键音 5  783.991HZ G5
localparam BTN6     = 56818;    //按键音 5  880HZ A6
localparam BTN7     = 50619;    //按键音 5  987.767HZ B6
localparam BTN8     = 47778;    //按键音 5  1046.502HZ C6
localparam FAN        = 1515151;//风扇噪音  33Hz

localparam DELAY      = 2500000;//时延 0.5s

//状态机
reg[21:0] beep_N = 22'b0;
reg[21:0] delay_N = DELAY;
always @(posedge sys_clk or negedge rst_n)begin
    if(!rst_n)begin
        beep_N <= 22'b0;
        delay_N <= DELAY;
    end else if(beep_N==FAN||delay_N==22'b0)begin
        case({key})
            8'b10000000: beep_N <= BTN1;
            8'b01000000: beep_N <= BTN2;
            8'b00100000: beep_N <= BTN3;
            8'b00010000: beep_N <= BTN4;
            8'b00001000: beep_N <= BTN5;
            8'b00000100: beep_N <= BTN6;
            8'b00000010: beep_N <= BTN7;
            8'b00000001: beep_N <= BTN8;
            default:         beep_N <= FAN;
        endcase
        delay_N <= key!=8'b0 ? DELAY : 22'b0;
    end else delay_N <= delay_N-1;
end

//控制输出
reg out = 1'b0;
reg[21:0]cnt = 22'b0;
always @(posedge sys_clk or negedge rst_n)begin
    if(!rst_n)beep <= 1'b0;
    else beep <= out;
    if(!rst_n)begin
```

```
        out <= 0;
        cnt <= 22'b0;
    end else if(cnt==beep_N-1||delay_N==DELAY)begin //当计数到达或开始延时
        out <= out+1;
        cnt <= 22'b0;
    end else cnt <= cnt+1;
end

endmodule
```

4.8 项目主文件(bath_heater.v)

```
/*
Author: James Hoi
Date: 2021.12.02
Description: 数电大实验实验一题目：浴霸
*/

module bath_heater(
    input clk,              //时钟为 50MHz
    input rst,              //SW6 重置电路
    input[7:0] btn,     //BTN7~BTN0
    input[3:0] keyboard_row,    //4x4 键盘行
    output[3:0] keyboard_col,   //4x4 键盘列
    output[15:0] led,       //灯 LD15~LD0
    output[7:0] led_r,  //红色行
    output[7:0] led_g,  //绿色行
    output[7:0] led_d,  //共地，控制列
    output[7:0] seg_led,    //数码管
    output[7:0] seg_cat,    //选择数码管
    output beep             //蜂鸣器
);

    wire clk_led;                       //扫描 led 时钟
    wire clk_ani;                       //动画时钟
    wire clk_seg;                       //扫描数码管时钟
    wire clk_keyboard;              //扫描 4x4 键盘时钟
    wire[63:0] matrix_r;                //当前状态动画红色矩阵值
    wire[63:0] matrix_g;                //当前状态动画蓝色矩阵值
    wire[7:0] key_pulse;                //消抖后的按键脉冲
    wire[15:0] keyboard_pulse;      //消抖后的 4x4 键盘脉冲
    wire[31:0] seg_data;                //数码管显示的值
    wire[7:0] seg_on;                   //数码管的开关
    wire[7:0] key = keyboard_pulse[7:0]|key_pulse; //4x4 键盘按键和正常按键


    //实例化按键蜂鸣器模块
    key_beep b3(
    .sys_clk(clk),
    .rst_n(rst),
    .key(key),
    .beep(beep)
    );


    //实例化按键消抖
    debounce #(8) d1(
```

```
.clk(clk),
.rst(rst),
.key(btn),
.key_pulse(key_pulse)
);


//实例化 4x4 键盘并消抖
keyboard k1(
.sys_clk(clk),
.rst_n(rst),
.row(keyboard_row),
.col(keyboard_col),
.pulse(keyboard_pulse)
);


//产生一个 1kHz 时钟信号，N=50000
//50000000/50000=1kHz 延时 1ms
//分频给数码管时钟
divide #(.WIDTH(32),.N(50000)) u1(
.clk(clk),
.rst_n(rst),
.clkout(clk_seg)
);
//例化数码管组
segment s1(
.clk(clk_seg),
.rst_n(rst),
.seg_data(seg_data),
.seg_led(seg_led),
.seg_on(seg_on),
.cat(seg_cat)
);


//产生一个 10kHz 时钟信号，N=5000
//50000000/5000=10kHz
//分频给 LED 扫描时钟
divide #(.WIDTH(32),.N(5000)) u2(
.clk(clk),
.rst_n(rst),
.clkout(clk_led)
);
//例化矩阵 LED
led_matrix l1(
.clk(clk_led),
.rst_n(rst),
.matrix_r(matrix_r),
.matrix_g(matrix_g),
.row(led_d),
.col_r(led_r),
.col_g(led_g),
);


//产生 2Hz 时钟信号,N=25000000
//仿真设置为 2MHz,N=25
```

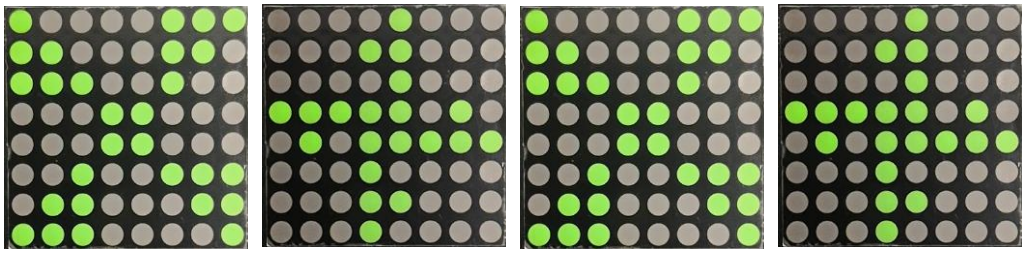
```
    //分频给动画时钟
    divide #(.WIDTH(32),.N(25000000)) u3(
    .clk(clk),
    .rst_n(rst),
    .clkout(clk_ani)
    );
    //例化 LED 动画模块
    animate a1(
    .sys_clk(clk),
    .clk_ani(clk_ani),
    .rst_n(rst),
    .key(key),
    .matrix_r(matrix_r),
    .matrix_g(matrix_g),
    .seg_data(seg_data),
    .seg_on(seg_on),
    .led(led)
    );

endmodule
```

第五章 功能说明及资源利用情况

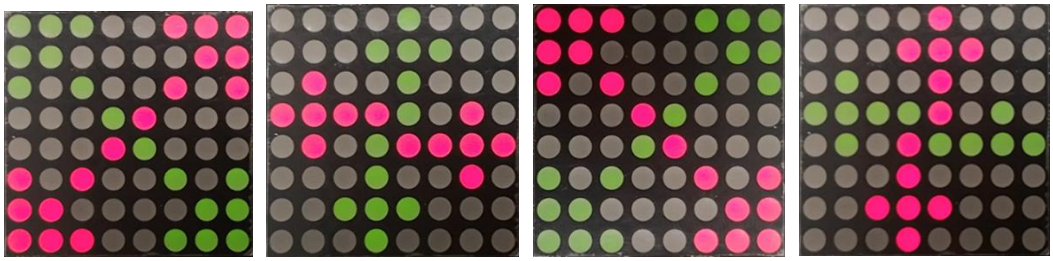
5.1 功能说明

5.1.1 换气模式点阵动画



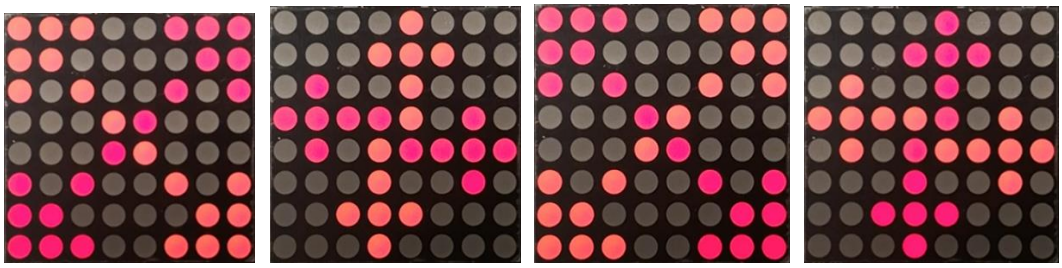
可见点阵显示与题目相符。

5.1.2 风暖模式点阵动画



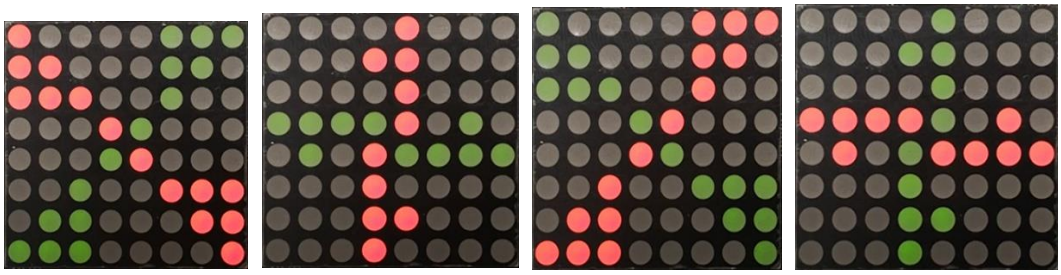
可见点阵显示与题目相符。

5.1.3 强暖模式点阵动画



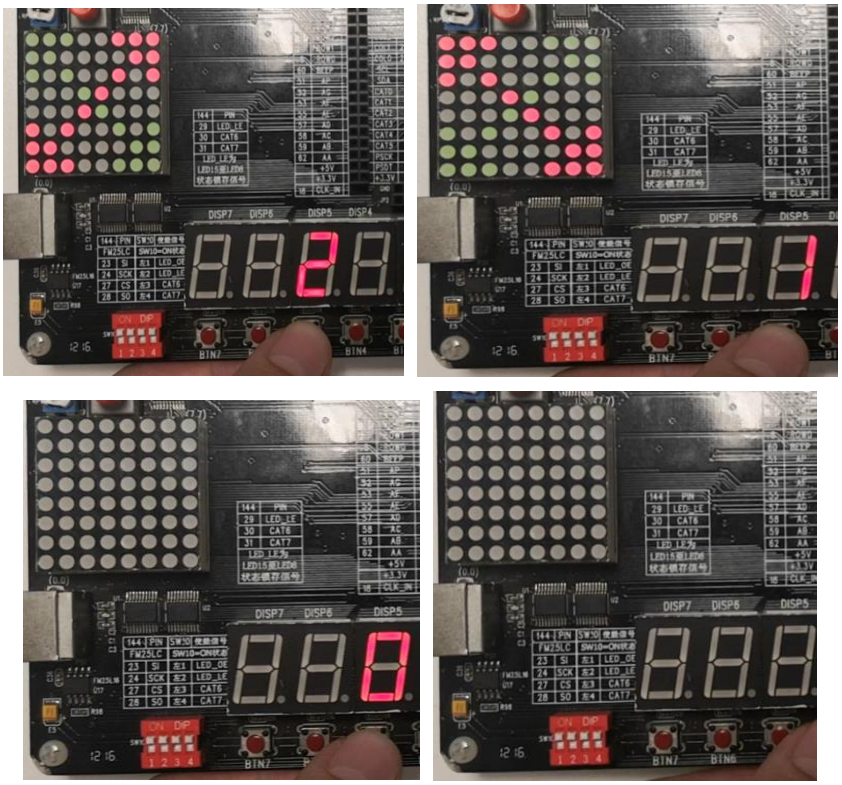
可见点阵显示与题目相符。

5.1.4 干燥模式点阵动画

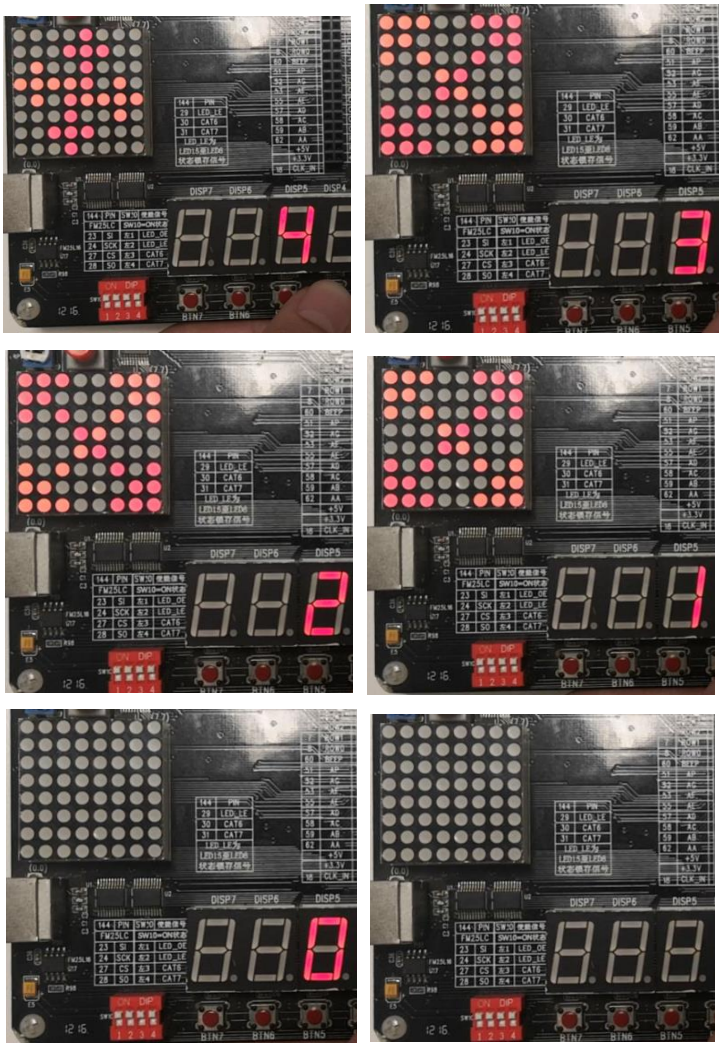


可见点阵显示与题目相符。

5.1.5 数码管显示延时秒数

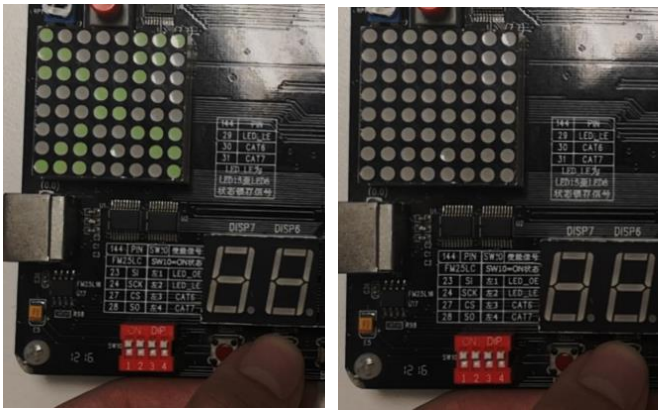


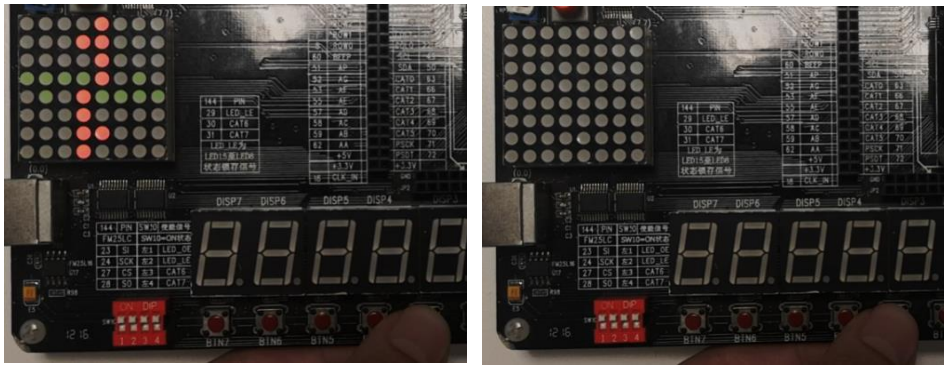
可见当模式为风暖模式时按下 BTN5，数码管显示延时秒数，延时 4 秒后停止动画，结果与题目相符。



可见当模式为强暖模式时按下 BTN4，数码管显示延时秒数，延时 2 秒后停止动画，结果与题目相符。

5.1.6 点阵立即停止动画

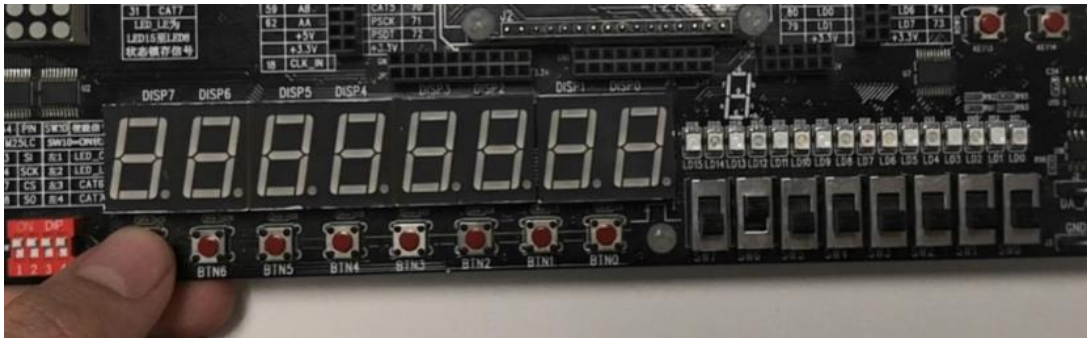




可见当模式为换气模式时按下 BTN6，或当模式为干燥模式时按下下 BTN3，动画立即停止，结果与题目相符。

5.1.7 照明灯切换

按下 BTN7 前,照明灯 LD6 关闭。



按下 BTN7 后,照明灯 LD6 开启，再按一次照明灯关闭。

5.2 资源利用情况

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Dec 17 00:22:32 2021
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	bath_heater
Top-level Entity Name	bath_heater
Family	MAX II
Device	EPM1270T144C5
Timing Models	Final
Total logic elements	621 / 1,270 (49 %)
Total pins	75 / 116 (65 %)
Total virtual pins	0
UFM blocks	0 / 1 (0 %)

本程序管脚占用为65%，逻辑单元利用情况为49%。

第六章 故障及问题分析

6.1 数码管扫描频率

编写数码管译码器时无法让数码管正常工作，经调试后发现是没有给对正确的工作时钟，调整为1kHz后数码管能够正常工作。

6.2 按键消抖

由于数电课每周都有布置Verilog实验，所以最开始按键消抖模块使用的是小脚丫FPGA的代码，该代码用于实验课的实验板无法正常工作。经调试，发现问题是由于实验板的按键电平与小脚丫的是相反的，修改按键消抖模块中的脉冲边沿检测即可，将下降沿检测修改为上升沿检测。

6.3 按键后无法立刻停止动画

一开始编写动画模块时，切换模式的状态机的时钟和动画切换的时钟频率是一样的，导致按键后需要等待0.5秒动画才能停止。将切换模式的状态机与动画切换的状态机分离，两者使用不同的时钟频率，其中切换模式使用系统时钟即可解决这个问题。

第七章 总结和结论

通过此次实验，我对于按键消抖的原理以及时钟分频有了更加深刻的理解，掌握了无源蜂鸣器、双色点阵、数码管控制的具体实现方法；对于一个较复杂系统功能实现的具体流程有了一个总体的认识和体会，了解并掌握了自顶向下的FPGA模块化设计，将一个大问题分解为若干个小问题逐个击破；对于模块功能的验证，由于一些功能仅通过硬件是难以发现错误的地方，需要利用仿真波形确认，我学会了利用仿真调试电路功能的方法。

这次实验是本学期所学数字电路的一次综合应用，不仅让我对于书本上的知识更加深刻，也大大的锻炼了我的学习能力。数字电路的调试相比任何一种计算机语言包括汇编的调试都更为复杂，因为无法步进判断每一语句且无法查看中间变量的值。有时，仿真波形显示程序能够正常工作，但烧录后却无法达到想要的效果，加上没有像计算机语言的编译优化，每次编译都十分费时，烧录也是同样。

总而言之，我相信这次的综合实验对于每一位同学都是一次全新的挑战，但收获一定是巨大的。最后感谢老师，感谢帮我解答疑问的同学，感谢这次实验！