# EDA 实验课程设计——电梯控制器

设计者：上海交通大学 信息安全工程学院 F0503602 石君霄 5050369043

2007-04-02 ~ 2007-04-21

## 设计目标

模拟上海交通大学闵行校区电子信息与电气工程学院 3 号楼内电梯的工作流程。

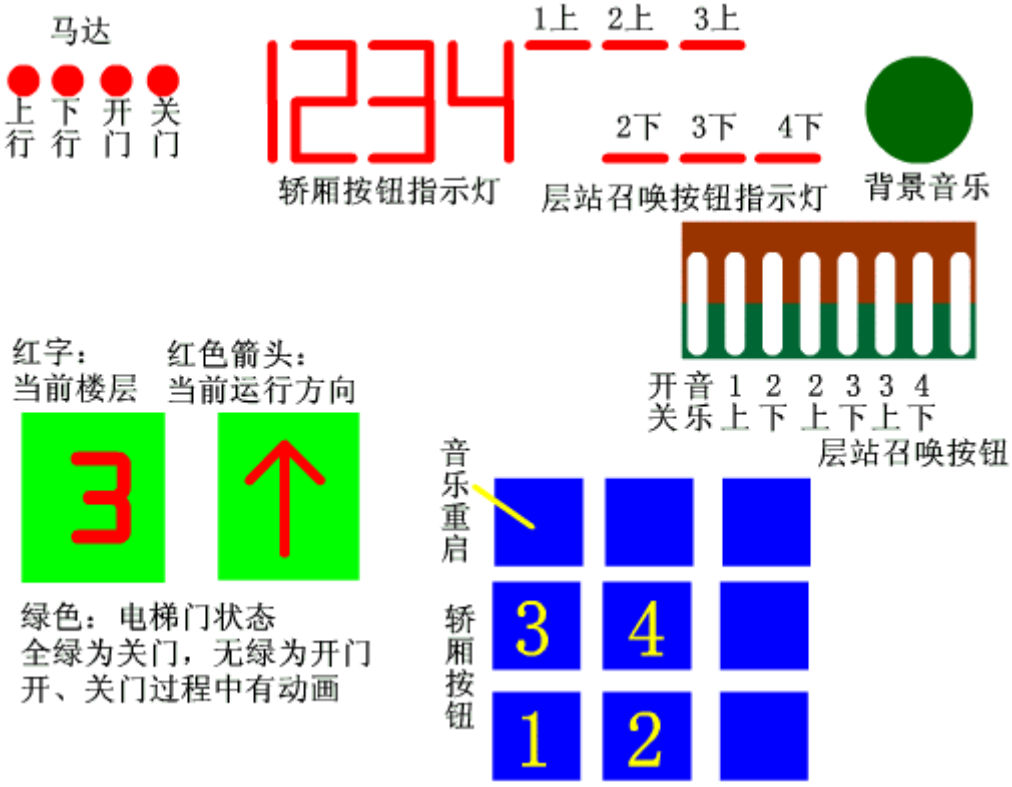
1. 4 个层站：1 楼、2 楼、3 楼、4 楼
2. 轿厢设有目的地选择按钮 4 个，按钮有指示灯
3. 层站设有召唤按钮：1 楼上，2 楼下、上，3 楼下、上，4 楼下，按钮有指示灯
4. 轿厢和层站设有当前位置、运行方向显示屏
5. 2 个马达：上/下运行，开/关门
6. 不支持开门、关门、紧急等其他按钮
7. 增设固定乐曲的背景音乐功能

## 实验箱输入、输出对应

使用带有 FLEX®EPF10K20TC144-4 芯片的 JDEE-10K 实验箱，实验箱上可用资源有：

- 输入：9 个微动按键开关，8 个拨码开关
- 输出：4 个 LED 指示灯，8 位七段带小数点数码管，2 个 8×8 红、绿双色点阵，1 个蜂鸣器
- 信号：2KHz 晶振，4MHz 晶振

具体对应关系如图所示



# 基本设计思想

全面采取分层次、分模块设计，控制、输入输出分离。

# 顶层图

顶层采用 GDF 图形设计，以便更直观的显示设计思想；请看附图。图中涉及组件介绍及代码见后面介绍。微动输入端后所加的 D 触发器，用于消除抖动。

# 控制组件 ctrl.tdf

**控制电梯的运行流程**

```
title "elevator control module";
include "lpm_counter";
subdesign ctrl
(
--clock for switching states, a new running or stopped state each 8 signals
运行状态变换的时钟，每 8 次脉冲电梯进入一个新的运转或停止状态
   clk:input;--频率应为 1Hz
--clock for button detection 较高的频率，用于检测按钮
   bclk:input;
--button on each floor for service requests 层站召唤按钮，1 为按下
   b1up,b2dn,b2up,b3dn,b3up,b4dn:input;
--indicator lights behind floor buttons 层站召唤按钮指示灯，1 为灯亮
   i1up,i2dn,i2up,i3dn,i3up,i4dn:output;
--button inside lifter to input target floor 轿厢按钮，1 为按下
   b1to,b2to,b3to,b4to:input;
--indicator lights behind lifter buttons 轿厢按钮指示灯，1 为灯亮
   i1to,i2to,i3to,i4to:output;
--current floor display (BCD) 当前楼层，用 BCD 码表示
   disp[3..0]:output;
--current floor display flashing? 当前楼层显示是否闪烁，1 为闪烁
   dispflash:output;
--current running direction: 0=up,1=dn 当前运行方向，0 为上行、1 为下行
   dispdir:output;
--up motor control 上行马达，1 为运转
   motorup:output;
--down motor control 下行马达，1 为运转
   motordn:output;
--door open motor control 开门马达，1 为运转
   motordooropen:output;
--door shut motor control 关门马达，1 为运转
   motordoorshut:output;
)
variable
--current stopped location or running state 当前运行状态，依次为：(停在 1 层，停在 2
```

**层，停在3层，停在4层，1层到2层中，2层到3层中，3层到4层中，4层到3层中，3层到2层中，2层到1层中)**

```
    s:machine                                                    with
states(stop1,stop2,stop3,stop4,1to2,2to3,3to4,4to3,3to2,2to1);
--current counter 状态切换等待计数器，计数满后切换状态
-- when stopped 停止时：
--  0~1=opening door 开门
--  2~5=door opened 门开着
--  6~7=shutting door 关门
    c:lpm_counter with(lpm_width=3,lpm_direction="UP");
--current running direction 当前运行方向，依次为：(上行，下行)
    dir:machine with states(up,dn);
--need the lifter goto and stop at those floors? 是否需要前往并停在这些层站?
    needgo[4..1]:node;
--is there a request, either from lifter buttons or from floor buttons, that
request the lifter to go up or go down? 是否有来自轿厢按钮或层站召唤按钮的请求而需要
上行/下行?
    needup,needdn:node;
--no request? remain stopped and have a break (1=stop) 没有请求吗？可以在此地休
息一下(1=可以休息)
    stophere:node;
--button pressed and not processed? 已经被按下过、但没有处理的按钮
    b1t,b1u,b2d,b2t,b2u,b3d,b3t,b3u,b4d,b4t:jkff;
begin
    defaults --默认值区段，参见下面相关内容
--needup,needdn logic 需要上行、需要下行逻辑
        needup=gnd;
        needdn=gnd;
--motors control output 马达控制输出
        motorup=gnd;
        motordn=gnd;
        motordooropen=gnd;
        motordoorshut=gnd;
--buttons and indicator lights  按钮和指示灯处理
        b1t.clrn=vcc;
        b1u.clrn=vcc;
        b2d.clrn=vcc;
        b2t.clrn=vcc;
        b2u.clrn=vcc;
        b3d.clrn=vcc;
        b3t.clrn=vcc;
        b3u.clrn=vcc;
        b4d.clrn=vcc;
        b4t.clrn=vcc;
        c.cnt_en=vcc;
    end defaults;
```

```
    c.clock=clk;
    s.clk=c.cout;
    dir.clk=c.cout;
```

**--buttons and indicator lights** 按钮和指示灯处理

**--使用 JK 触发器，按钮输入接到 J 端、按钮按下时置位，到达所要求的层站时使用 clrn 直接复位(离开这个层站前按钮再次被按下无效)，Q 端接指示灯。不能使用 J/K 两端，否则停在该层站时按钮被按下，将导致触发器 Toggle 翻转，结果不正确；更不能使用 prn/clrn 两端，否则停在该层站时按钮被按下，将导致触发器出错；因此应该选择 J/clrn 两端。注意 clrn 必须在 defaults 中接高电平，否则会不正常。**

```
--turn off indicator lights on next state by setting .ena to vcc
--turn off indicator lights immediately by setting .clrn to gnd
    b1t.j=b1to;b1t.k=gnd;b1t.clk=bclk;i1to=b1t.q;
    b1u.j=b1up;b1u.k=gnd;b1u.clk=bclk;i1up=b1u.q;
    b2d.j=b2dn;b2d.k=gnd;b2d.clk=bclk;i2dn=b2d.q;
    b2t.j=b2to;b2t.k=gnd;b2t.clk=bclk;i2to=b2t.q;
    b2u.j=b2up;b2u.k=gnd;b2u.clk=bclk;i2up=b2u.q;
    b3d.j=b3dn;b3d.k=gnd;b3d.clk=bclk;i3dn=b3d.q;
    b3t.j=b3to;b3t.k=gnd;b3t.clk=bclk;i3to=b3t.q;
    b3u.j=b3up;b3u.k=gnd;b3u.clk=bclk;i3up=b3u.q;
    b4d.j=b4dn;b4d.k=gnd;b4d.clk=bclk;i4dn=b4d.q;
    b4t.j=b4to;b4t.k=gnd;b4t.clk=bclk;i4to=b4t.q;
```

**--determine where to go if the lifter is running** 当电梯正在运转时，确定下一个状态

```
    table
        s    ,needgo1,needgo2,needgo3,needgo4=>s    ,dir;
        1to2 ,   x  ,   1  ,   x  ,   x  =>stop2, up;--在 1 层至 2 层间，需要停 2 层，则停
        1to2 ,   x  ,   0  ,   x  ,   x  =>2to3 , up;--在 1 层至 2 层间，不需要停 2 层，则继续上行(不可能出现不需要继续上行的情况，那样就不可能开始向 2 层运行)
        2to3 ,   x  ,   x  ,   1  ,   x  =>stop3, up;--与上面类似，不再说明
        2to3 ,   x  ,   x  ,   0  ,   x  =>3to4 , up;
        3to4 ,   x  ,   x  ,   x  ,   x  =>stop4, dn;
        4to3 ,   x  ,   x  ,   1  ,   x  =>stop3, dn;
        4to3 ,   x  ,   x  ,   0  ,   x  =>3to2 , dn;
        3to2 ,   x  ,   1  ,   x  ,   x  =>stop2, dn;
        3to2 ,   x  ,   0  ,   x  ,   x  =>2to1 , dn;
        2to1 ,   x  ,   x  ,   x  ,   x  =>stop1, up;
    end table;
```

**--turn off inside button indicator lights when arrive** 到达时，熄灭轿厢按钮指示灯

```
    case s is
        when stop1=>
            b1t.clrn=gnd;
        when stop2=>
            b2t.clrn=gnd;
```

```
        when stop3=>
            b3t.clrn=gnd;
        when stop4=>
            b4t.clrn=gnd;
    end case;
```

**--can have a rest? stop counter to avoid opening and shutting door again and again** 休息逻辑，如果不需要上行也不需要下行，将状态切换等待计数器停在 4 位置，避免反复开门、关门

```
    stophere=!(needup # needdn);
    if stophere then
        if c.q[]==4 then
            c.cnt_en=gnd;
        end if;
```

**--determine where to go when the lifter is stopped** 决定停止时收到请求，电梯向何处运行

**--turn off outside button indicator light when arrive and same direction** 当到达层站且方向相同时，熄灭层站召唤按钮指示灯

```
    else
```

**--going up, and a request is upstairs** 正在上行，且上面有请求

**--or going down, but no request is downstairs, however there is a request (must be upstairs), should go up** 或正在下行，但是下面没有请求(即上面有请求)，应该转为上行

```
        if (dir==up & needup) # (dir==dn & !needdn) then
            dir=up;
            case s is
                when stop1=>
                    s=1to2;
                    b1u.clrn=gnd;
                when stop2=>
                    s=2to3;
                    b2u.clrn=gnd;
                when stop3=>
                    s=3to4;
                    b3u.clrn=gnd;
            end case;
        end if;
--going down, and a request is downstairs  与上行情况相反，不再说明
--or going up, but no request is upstairs, however there is a request (must
be downstairs), should go down
        if (dir==dn & needdn) # (dir==up & !needup) then
            dir=dn;
            case s is
                when stop4=>
                    s=4to3;
                    b4d.clrn=gnd;
```

```
            when stop3=>
                s=3to2;
                b3d.clrn=gnd;
            when stop2=>
                s=2to1;
                b2d.clrn=gnd;
        end case;
    end if;
end if;

--display output  显示输出
    table --当前楼层显示
        s    =>disp[];
        stop1=>1;
        stop2=>2;
        stop3=>3;
        stop4=>4;
        1to2 =>1;
        2to3 =>2;
        3to4 =>3;
        4to3 =>4;
        3to2 =>3;
        2to1 =>2;
    end table;
    if (s==stop1 # s==stop2 # s==stop3 # s==stop4) & (c.q[]==0 # c.q[]==1)
then
        dispflash=vcc; --停站开门时，当前楼层显示闪烁
    else
        dispflash=gnd;
    end if;
    table --运行方向显示
        dir=>dispdir;
        up =>   0;
        dn =>   1;
    end table;
```

**--needgo logic "是否需要去？"逻辑,如果该层站的召唤按钮被按下、或轿厢去该层站的按钮被按下，就需要去(并停站)**

```
    needgo[1]=b1u.q # b1t.q;
    needgo[2]=b2d.q # b2t.q # b2u.q;
    needgo[3]=b3d.q # b3t.q # b3u.q;
    needgo[4]=b4d.q # b4t.q;
```

**--needup,needdn logic "是否需要上行/下行？"逻辑**

**--(also some in defaults statement)** 默认区段中规定默认值为不需要

**if (s==stop1 # s==1to2 # s==2to1) & (needgo[2] # needgo[3] # needgo[4]) then needup=vcc; end if;--**如果当前低于 **2** 层，需要去 **2**、**3**、**4** 层，则需要上行；以下类同

```
    if (s==stop2 # s==2to3 # s==3to2) & (needgo[3] # needgo[4]) then
needup=vcc; end if;
    if (s==stop3 # s==3to4 # s==4to3) & (needgo[4]) then needup=vcc; end if;
    if (s==stop4 # s==4to3 # s==3to4) & (needgo[1] # needgo[2] # needgo[3])
then needdn=vcc; end if;
    if (s==stop3 # s==3to2 # s==2to3) & (needgo[1] # needgo[2]) then
needdn=vcc; end if;
    if (s==stop2 # s==2to1 # s==1to2) & (needgo[1]) then needdn=vcc; end if;

--motors control output 马达控制输出
--(also some in defaults statement) 默认区段中规定默认值为马达不运转
    if (s==1to2 # s==2to3 # s==3to4) then motorup=vcc; end if; --上行时，启动上
行马达
    if (s==4to3 # s==3to2 # s==2to1) then motordn=vcc; end if; --下行时，启动下
行马达
    if (s==stop1 # s==stop2 # s==stop3 # s==stop4) then --停止时状态切换等待计数
器的值表示开门、关门
        if (c.q[]==0 # c.q[]==1) then motordooropen=vcc; end if;
        if (c.q[]==6 # c.q[]==7) then motordoorshut=vcc; end if;
    end if;
end;
```

# 显示组件 display.tdf

**负责电梯上的各项显示。控制组件只给出了“显示什么”，本组件需要给出“如何显示”，例如具体的字模、笔画、开/关门动画等。**

```
title "elevator display module";
include "lpm_counter";
subdesign display
(
--indicator lights behind floor buttons 层站召唤按钮指示灯
    i1up,i2dn,i2up,i3dn,i3up,i4dn:input;
--indicator lights behind lifter buttons 轿厢按钮指示灯
    i1to,i2to,i3to,i4to:input;
--current floor display (BCD) 当前楼层 BCD
    disp[3..0]:input;
--current floor display flashing? 当前楼层数字是否闪烁，1=闪烁
    dispflash:input;
--current running direction: 0=up,1=dn 当前运行方向，0=上行，1=下行
    dispdir:input;
--up motor control 上行马达
    motorup:input;
--down motor control 下行马达
    motordn:input;
--door open motor control 开门马达
```

```
    motordooropen:input;
--door shut motor control 关门马达
    motordoorshut:input;

--outputs
    led1,led2,led3,led4:output; --LED 输出，直接接到共阳接法的 LED
    digit1[3..0],digit2[3..0],digit3[3..0],digit4[3..0],digit5[3..0],digit6[3 .
.0],digit7[3..0],digit8[3..0]:output; --数码管输出，接到数码管组件
    1red1[7..0],1red2[7..0],1red3[7..0],1red4[7..0],1red5[7..0],1red6[7..0],1
red7[7..0],1red8[7..0],2red1[7..0],2red2[7..0],2red3[7..0],2red4[7..0],2red5
[7..0],2red6[7..0],2red7[7..0],2red8[7..0],1green1[7..0],1green2[7..0],1gree
n3[7..0],1green4[7..0],1green5[7..0],1green6[7..0],1green7[7..0],1green8[7..
0],2green1[7..0],2green2[7..0],2green3[7..0],2green4[7..0],2green5[7..0],2gr
een6[7..0],2green7[7..0],2green8[7..0]:output; --点阵字模输出，接到点阵组件

--2k clock 2KHz 时钟信号，用于数字闪烁、开/关门动画等
    clk:input
)
variable
--flashing freq=2Hz 闪烁，频率为 2Hz
    c_flash:lpm_counter with(lpm_width=10,lpm_direction="UP");
    flash:tff; --这个 T 触发器控制闪烁，.q=1 时亮，.q=0 时灭
--door open/close animation 开/关门动画所需的 4Hz 频率、门框位置计数器、门状态等
    c_ani:lpm_counter with(lpm_width=9,lpm_direction="UP");--4Hz
    ani:lpm_counter with(lpm_width=3,lpm_svalue=7);
    doorstatus:machine with states(shut,opening,open,shutting);
    ani_l[7..0],ani_r[7..0]:node; --左、右点阵绿色列
begin
    defaults --默认使点阵红色熄灭、绿色亮起(门框、门眉)
        1red1[]=gnd;
        1red2[]=gnd;
        1red3[]=gnd;
        1red4[]=gnd;
        1red5[]=gnd;
        1red6[]=gnd;
        1red7[]=gnd;
        1red8[]=gnd;
        2red1[]=gnd;
        2red2[]=gnd;
        2red3[]=gnd;
        2red4[]=gnd;
        2red5[]=gnd;
        2red6[]=gnd;
        2red7[]=gnd;
        2red8[]=gnd;
        1green1[]=vcc;
```

```
        1green2[]=vcc;
        1green3[]=vcc;
        1green4[]=vcc;
        1green5[]=vcc;
        1green6[]=vcc;
        1green7[]=vcc;
        1green8[]=vcc;
        2green1[]=vcc;
        2green2[]=vcc;
        2green3[]=vcc;
        2green4[]=vcc;
        2green5[]=vcc;
        2green6[]=vcc;
        2green7[]=vcc;
        2green8[]=vcc;
    end defaults;
```

**--motors => LEDs** 马达状态显示在 **LED**，共阳接法 **LED** 需要反相

```
    led1=!motorup;
    led2=!motordn;
    led3=!motordooropen;
    led4=!motordoorshut;
```

**--lifter indicators => 1~4 digits** 数码管 **1~4** 位作为轿厢按钮指示灯；译码器中将 **0H** 作为空白显示，因此输出 **0** 数码管熄灭

```
    digit1[]=i1to & 1;
    digit2[]=i2to & 2;
    digit3[]=i3to & 3;
    digit4[]=i4to & 4;
```

**--floor indicators => 5~8 digits** 数码管 **5~8** 位作为层站召唤按钮指示灯；译码器中将 **CH**、**DH**、**EH**、**FH** 作为空白、下灯、上灯、下灯和上灯显示

```
    digit53=vcc;digit52=vcc;digit51=i1up;digit50=gnd;
    digit63=vcc;digit62=vcc;digit61=i2up;digit60=i2dn;
    digit73=vcc;digit72=vcc;digit71=i3up;digit70=i3dn;
    digit83=vcc;digit82=vcc;digit81=gnd;digit80=i4dn;
```

**--current floor => dot1 red** 当前楼层显示在点阵 **1** 红色

```
    c_flash.clock=clk;
    flash.clk=clk;
    flash.t=c_flash.cout;
    if !dispflash # flash.q then --闪烁、灯灭时，按 defaults 中默认值输出：灭
        case disp[] is --以下为 1~4 数字字模
            when 1=>
                1red1[]=B"00000000";
                1red2[]=B"00001000";
```

```
            1red3[]=B"00011000";
            1red4[]=B"00001000";
            1red5[]=B"00001000";
            1red6[]=B"00001000";
            1red7[]=B"00011100";
            1red8[]=B"00000000";
        when 2=>
            1red1[]=B"00000000";
            1red2[]=B"00011100";
            1red3[]=B"00100010";
            1red4[]=B"00000100";
            1red5[]=B"00001000";
            1red6[]=B"00010000";
            1red7[]=B"00111110";
            1red8[]=B"00000000";
        when 3=>
            1red1[]=B"00000000";
            1red2[]=B"00111100";
            1red3[]=B"00000010";
            1red4[]=B"00000100";
            1red5[]=B"00000100";
            1red6[]=B"00000010";
            1red7[]=B"00111100";
            1red8[]=B"00000000";
        when 4=>
            1red1[]=B"00000100";
            1red2[]=B"00001100";
            1red3[]=B"00010100";
            1red4[]=B"00100100";
            1red5[]=B"01000100";
            1red6[]=B"01111110";
            1red7[]=B"00000100";
            1red8[]=B"00001110";
    end case;
end if;
```

**--direction => dot2 red** 运行方向用箭头显示在点阵 **2** 红色

```
if dispdir then--dn 向下箭头字模
    2red2[]=B"00010000";
    2red3[]=B"00010000";
    2red4[]=B"00010000";
    2red5[]=B"01010100";
    2red6[]=B"01010100";
    2red7[]=B"00111000";
else--up 向上箭头字模
    2red2[]=B"00111000";
```

```
        2red3[]=B"01010100";
        2red4[]=B"01010100";
        2red5[]=B"00010000";
        2red6[]=B"00010000";
        2red7[]=B"00010000";
    end if;
```

**--door open/close animation => dot1&2 green** 开/关门状态及过程动画显示在点阵绿色

```
    c_ani.clock=clk;
    ani.clock=clk;
    ani.clk_en=c_ani.cout; --动画计数器，向下计数为开门，向上计数为关门
    doorstatus.clk=clk;
    doorstatus.ena=c_ani.cout;
    table --马达导致的门状态转换表

    doorstatus,motordooropen,motordoorshut=>doorstatus,ani.updown,ani.cnt_en,
ani.sclr,ani.sset;
        shut     ,     1       ,     x       => opening  ,   1      ,   1      ,
1    ,   0    ;
        opening  ,     0       ,     x       => open     ,   1      ,   0      ,
0    ,   0    ;
        open     ,     x       ,     1       => shutting ,   0      ,   1      ,
0    ,   1    ;
        shutting ,     x       ,     0       => shut     ,   0      ,   0      ,
0    ,   0    ;
    end table;
    if doorstatus==open # doorstatus==shut then
        table --开门、关门过程中，显示动画；ani 是动画计数器
            ani.q[] => ani_l[];
              7    => B"11111111";
              6    => B"11111110";
              5    => B"11111100";
              4    => B"11111000";
              3    => B"11110000";
              2    => B"11100000";
              1    => B"11000000";
              0    => B"10000000";
        end table;
        ani_r7=ani_l0; --右边点阵显示正好与左边相反
        ani_r6=ani_l1;
        ani_r5=ani_l2;
        ani_r4=ani_l3;
        ani_r3=ani_l4;
        ani_r2=ani_l5;
        ani_r1=ani_l6;
        ani_r0=ani_l7;
```

```
        1green2[]=ani_l[];2green2[]=ani_r[]; --defaults 规定 1、8 行常亮
        1green3[]=ani_l[];2green3[]=ani_r[]; --2~7 行显示动画，内容相同
        1green4[]=ani_l[];2green4[]=ani_r[];
        1green5[]=ani_l[];2green5[]=ani_r[];
        1green6[]=ani_l[];2green6[]=ani_r[];
        1green7[]=ani_l[];2green7[]=ani_r[];
    end if;
end;
```

# 背景音乐组件 music.tdf

**用于播放背景音乐，实际上与电梯的主体并无任何联系，只是一个娱乐性功能。**

```
--playing <Song Of Joy> 播放曲目:《欢乐颂》贝多芬
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  3  2  2  -  |
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  2  1  1  -  |
-- 2  2  3  1  |  2  34 3  1  |  2  34 3  2  |  1  2  5  -  |
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  2  1  1  -  |
--注：程序按以上简谱实现，但是简谱中有几个音符错误，正确的简谱是：
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  3  -  2  2  |
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  2  -  1  1  |
-- 2  2  3  1  |  2  34 3  1  |  2  34 3  2  |  1  2  5  -  |
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  2  -  1  1  |

include "lpm_counter";
subdesign music
(
--4MHz freq from OSC 4MHz 晶振信号
    clk:input;
--enable, no output when 0 音乐开关，0 时无输出
    ena:input;
--clear, return to beginning on 1 重启信号，高电平是音乐从头开始
    clr:input;
--speaker 蜂鸣器输出
    spk:output;
)
variable
--freq necessary, 391.995Hz,523.25Hz,587.25Hz,659.25Hz,698.46Hz,783.99Hz
    L5,M1,M2,M3,M4,M5:node; --乐曲中需要的声音频率：5 1 2 3 4 5
--counters for freq generate 用计数器产生声音频率
    --cL5 is unnecessary because L5 is half of M5 5 不需要计数器，因为 5 是 5 的一半，
可以分频得到
    cM1:lpm_counter with(lpm_width=13,lpm_direction="UP",lpm_modulus=7645);
    cM2:lpm_counter with(lpm_width=13,lpm_direction="UP",lpm_modulus=6810);
    cM3:lpm_counter with(lpm_width=13,lpm_direction="UP",lpm_modulus=6068);
    cM4:lpm_counter with(lpm_width=13,lpm_direction="UP",lpm_modulus=5727);
    cM5:lpm_counter with(lpm_width=13,lpm_direction="UP",lpm_modulus=5102);
```

```
--tff for freq generate 计数器.cout 输出的声音频率信号占空比很小，用 T 触发器获得占空比 1/2 的声音频率信号
    tL5,tM1,tM2,tM3,tM4,tM5:tff;
--main counter for music sequence 音乐序列主计数器
-- [8..7]=line 乐谱行
-- [6..5]=group 小节
-- [4..3]=full note 拍
-- [2..0]=1/8 note 1/8 拍
    c:lpm_counter with(lpm_width=9,lpm_direction="UP");
--16Hz helper for music sequence
    divider:lpm_counter with(lpm_width=18,lpm_direction="UP");
    divd:dff;
--speaker, don't care enable input, don't care silent on c.q[2..0]==7 on non 'X-' note 输出信号，但不考虑音乐开关，也不考虑每个 1 拍或更长音符的最后 1/8 拍暂停
    s:node;
begin
    defaults
        s=gnd;
    end defaults;
--freq generate 产生声音频率
    cM1.clock=clk;
    cM2.clock=clk;
    cM3.clock=clk;
    cM4.clock=clk;
    cM5.clock=clk;
    tL5.t=vcc;
    tM1.t=cM1.cout;
    tM2.t=cM2.cout;
    tM3.t=cM3.cout;
    tM4.t=cM4.cout;
    tM5.t=cM5.cout;
    tL5.clk=tM5.q;
    tM1.clk=clk;
    tM2.clk=clk;
    tM3.clk=clk;
    tM4.clk=clk;
    tM5.clk=clk;
    L5=tL5.q;
    M1=tM1.q;
    M2=tM2.q;
    M3=tM3.q;
    M4=tM4.q;
    M5=tM5.q;
--music sequence
    divider.clock=clk;
    divd.clk=clk;
```

```
    divd.d=divider.cout;
    c.clock=divd.q;
-- line1,group1~3;line2;line4 第 1 行 1~3 小节；第 2、4 行
-- 3  3  4  5  |  5  4  3  2  |  1  1  2  3  |  2  1  1  -  |
    if c.q[7]==1 # (c.q[8]==0 & (c.q[6]==0 # c.q[5]==0)) then
        case c.q[6..3] is
            when  0=> s=M3;
            when  1=> s=M3;
            when  2=> s=M4;
            when  3=> s=M5;
            when  4=> s=M5;
            when  5=> s=M4;
            when  6=> s=M3;
            when  7=> s=M2;
            when  8=> s=M1;
            when  9=> s=M1;
            when 10=> s=M2;
            when 11=> s=M3;
            when 12=> s=M2;
            when 13=> s=M1;
            when 14=> s=M1;
            when 15=> s=M1;
        end case;
    end if;
-- line1,group4 第 1 行 4 小节
-- 3  2  2  -  |
    if c.q[8..7]==0 & c.q[6..5]==3 then
        case c.q[4..3] is
            when  0=> s=M3;
            when  1=> s=M2;
            when  2=> s=M2;
            when  3=> s=M2;
        end case;
    end if;
-- line3 第 3 行
-- 2  2  3  1  |  2  34 3  1  |  2  34 3  2  |  1  2  5  -  |
--               --           --              ^
    if c.q[8..7]==2 then
        case c.q[6..3] is
            when  0=> s=M2;
            when  1=> s=M2;
            when  2=> s=M3;
            when  3=> s=M1;
            when  4=> s=M2;
            when  5=>
                if c.q[2] then s=M4; end if;
```

```
            if !c.q[2] & !(c.q[1..0]==3) then s=M3; end if;
         when  6=> s=M3;
         when  7=> s=M1;
         when  8=> s=M2;
         when  9=>
            if c.q[2] then s=M4; end if;
            if !c.q[2] & !(c.q[1..0]==3) then s=M3; end if;
         when 10=> s=M3;
         when 11=> s=M2;
         when 12=> s=M1;
         when 13=> s=M2;
         when 14=> s=L5;
         when 15=> s=L5;
      end case;
   end if;
--speaker pause and enable 暂停、开关处理
   if ena then
      if !(c.q[2..0]==7) # c.q[6..3]==14 then
         spk=s;
      end if;
   end if;
--clear 复位处理，使音乐序列计数器复位即可
   c.sclr=clr;
end;
```

**--本组件音乐序列直接写入逻辑代码，缺乏通用性，可考虑利用 lpm_rom 创建通用的音乐组件**

# 1Hz 运行频率生成组件 1Hz.tdf

该组件使用 Megafunction lpm_counter 生成，参数为 LPM_WIDTH=11,LPM_DIRECTION="UP"，因为将 2KHz(2000Hz)晶振信号当作 2048Hz 处理，实际输出频率略小于 1Hz

# 数码管组件 7seg8digit.tdf

将 8 位十六进制数显示在数码管上，调用七段译码器，本身负责扫描算法、在不同时刻将不同数字送入译码器。

```
include "lpm_counter";
include "7seg";
subdesign 7seg8digit
(
   d7[3..0],d6[3..0],d5[3..0],d4[3..0],d3[3..0],d2[3..0],d1[3..0],d0[3..0]:i
nput; --要显示的 8 个数字
   clk:input; --时钟信号，较高的频率
   a,b,c,d,e,f,g,t1,t2,t3,t4,t5,t6,t7,t8:output; --直接接到数码管上
)
variable
   cnt:lpm_counter with(lpm_width=6,lpm_direction="UP"); --扫描计数器
   7s:7seg; --七段译码器
```

```
begin
    defaults --默认情况下，所有数码管都未被选中
        t1=gnd;
        t2=gnd;
        t3=gnd;
        t4=gnd;
        t5=gnd;
        t6=gnd;
        t7=gnd;
        t8=gnd;
    end defaults;
    cnt.clock=clk;
    a=7s.a;
    b=7s.b;
    c=7s.c;
    d=7s.d;
    e=7s.e;
    f=7s.f;
    g=7s.g;
    if !(cnt.q[2..0]==7) then --在扫描中提供 1/8 时间的间隔，防止拖影现象
    --测试时，发现如果不加以上判断，可能导致旁边的位置出现淡淡的拖影
        case cnt.q[5..3] is --根据计数器数值，依次将各位数字送入译码器，并选中相应位
            when 0=> 7s.hex[]=d0[]; t8=vcc;
            when 1=> 7s.hex[]=d1[]; t7=vcc;
            when 2=> 7s.hex[]=d2[]; t6=vcc;
            when 3=> 7s.hex[]=d3[]; t5=vcc;
            when 4=> 7s.hex[]=d4[]; t4=vcc;
            when 5=> 7s.hex[]=d5[]; t3=vcc;
            when 6=> 7s.hex[]=d6[]; t2=vcc;
            when 7=> 7s.hex[]=d7[]; t1=vcc;
        end case;
    end if;
end;
```

# 七段译码器组件 7seg.tdf

将 16 进制数转换成相应的七段码，为了适应电梯显示的需要，去除了无用数码，而将部分数码对应层站召唤按钮指示灯显示。

```
title "elevator 7-segment decoder";
--this is not a general purpose HEX decoder
subdesign 7seg
(
hex[3..0]:input;
a,b,c,d,e,f,g:output;
)
begin
```

```
table
hex[3..0]=>a,b,c,d,e,f,g;
       0=>1,1,1,1,1,1,1;--no display 空白
       1=>1,0,0,1,1,1,1;--floor 1 数字 1，用于轿厢按钮指示灯
       2=>0,0,1,0,0,1,0;--floor 2 数字 2，用于轿厢按钮指示灯
       3=>0,0,0,0,1,1,0;--floor 3 数字 3，用于轿厢按钮指示灯
       4=>1,0,0,1,1,0,0;--floor 4 数字 4，用于轿厢按钮指示灯
      12=>1,1,1,1,1,1,1;--no display 空白
      13=>1,1,1,0,1,1,1;--dn 层站按钮指示灯：下
      14=>0,1,1,1,1,1,1;--up 层站按钮指示灯：上
      15=>0,1,1,0,1,1,1;--dn&up 层站按钮指示灯：下和上
end table;
end;
```

# 点阵组件 dot.tdf

接受两块点阵、两种颜色按行排列的字模输入，负责所有的扫描算法，显示的点阵上。

```
title "dot matrix display";
include "lpm_counter";
subdesign dot
(
    1red1[7..0]:input=gnd; --点阵 1 红色字模按行排列
    1red2[7..0]:input=gnd;
    1red3[7..0]:input=gnd;
    1red4[7..0]:input=gnd;
    1red5[7..0]:input=gnd;
    1red6[7..0]:input=gnd;
    1red7[7..0]:input=gnd;
    1red8[7..0]:input=gnd;
    2red1[7..0]:input=gnd; --点阵 2 红色字模按行排列
    2red2[7..0]:input=gnd;
    2red3[7..0]:input=gnd;
    2red4[7..0]:input=gnd;
    2red5[7..0]:input=gnd;
    2red6[7..0]:input=gnd;
    2red7[7..0]:input=gnd;
    2red8[7..0]:input=gnd;
    1green1[7..0]:input=gnd; --点阵 1 绿色字模按行排列
    1green2[7..0]:input=gnd;
    1green3[7..0]:input=gnd;
    1green4[7..0]:input=gnd;
    1green5[7..0]:input=gnd;
    1green6[7..0]:input=gnd;
    1green7[7..0]:input=gnd;
    1green8[7..0]:input=gnd;
    2green1[7..0]:input=gnd; --点阵 2 绿色字模按行排列
```

```
    2green2[7..0]:input=gnd;
    2green3[7..0]:input=gnd;
    2green4[7..0]:input=gnd;
    2green5[7..0]:input=gnd;
    2green6[7..0]:input=gnd;
    2green7[7..0]:input=gnd;
    2green8[7..0]:input=gnd;
    clk:input; --时钟信号，较高频率
    row1:output; --行控制线
    row2:output;
    row3:output;
    row4:output;
    row5:output;
    row6:output;
    row7:output;
    row8:output;
    1r1:output; --点阵 1 红色列控制线
    1r2:output;
    1r3:output;
    1r4:output;
    1r5:output;
    1r6:output;
    1r7:output;
    1r8:output;
    1g1:output; --点阵 1 绿色列控制线
    1g2:output;
    1g3:output;
    1g4:output;
    1g5:output;
    1g6:output;
    1g7:output;
    1g8:output;
    2r1:output; --点阵 2 红色列控制线
    2r2:output;
    2r3:output;
    2r4:output;
    2r5:output;
    2r6:output;
    2r7:output;
    2r8:output;
    2g1:output; --点阵 2 绿色列控制线
    2g2:output;
    2g3:output;
    2g4:output;
    2g5:output;
    2g6:output;
```

```
    2g7:output;
    2g8:output;
)
variable
    count:lpm_counter with(lpm_width=3,lpm_direction="UP"); --扫描计数器
begin
    count.clock=clk;
```

**--依次选中每一行，在列控制线上加上适当的信号；以下这段代码使用 EXCEL 编写公式生成的**

```
if count.q[]==0 then    row1=vcc; row2=gnd; row3=gnd; row4=gnd; row5=gnd;
    row6=gnd; row7=gnd; row8=gnd; 1r1=!1red17; 1r2=!1red16; 1r3=!1red15;
    1r4=!1red14; 1r5=!1red13; 1r6=!1red12; 1r7=!1red11; 1r8=!1red10;
    1g1=!1green17;   1g2=!1green16;   1g3=!1green15;   1g4=!1green14;
    1g5=!1green13;   1g6=!1green12;   1g7=!1green11;   1g8=!1green10;
    2r1=!2red17; 2r2=!2red16; 2r3=!2red15; 2r4=!2red14; 2r5=!2red13;
    2r6=!2red12; 2r7=!2red11; 2r8=!2red10; 2g1=!2green17;   2g2=!2green16;
    2g3=!2green15;   2g4=!2green14;   2g5=!2green13;   2g6=!2green12;
    2g7=!2green11;   2g8=!2green10;   end if;
if count.q[]==1 then    row1=gnd; row2=vcc; row3=gnd; row4=gnd; row5=gnd;
    row6=gnd; row7=gnd; row8=gnd; 1r1=!1red27; 1r2=!1red26; 1r3=!1red25;
    1r4=!1red24; 1r5=!1red23; 1r6=!1red22; 1r7=!1red21; 1r8=!1red20;
    1g1=!1green27;   1g2=!1green26;   1g3=!1green25;   1g4=!1green24;
    1g5=!1green23;   1g6=!1green22;   1g7=!1green21;   1g8=!1green20;
    2r1=!2red27; 2r2=!2red26; 2r3=!2red25; 2r4=!2red24; 2r5=!2red23;
    2r6=!2red22; 2r7=!2red21; 2r8=!2red20; 2g1=!2green27;   2g2=!2green26;
    2g3=!2green25;   2g4=!2green24;   2g5=!2green23;   2g6=!2green22;
    2g7=!2green21;   2g8=!2green20;   end if;
if count.q[]==2 then    row1=gnd; row2=gnd; row3=vcc; row4=gnd; row5=gnd;
    row6=gnd; row7=gnd; row8=gnd; 1r1=!1red37; 1r2=!1red36; 1r3=!1red35;
    1r4=!1red34; 1r5=!1red33; 1r6=!1red32; 1r7=!1red31; 1r8=!1red30;
    1g1=!1green37;   1g2=!1green36;   1g3=!1green35;   1g4=!1green34;
    1g5=!1green33;   1g6=!1green32;   1g7=!1green31;   1g8=!1green30;
    2r1=!2red37; 2r2=!2red36; 2r3=!2red35; 2r4=!2red34; 2r5=!2red33;
    2r6=!2red32; 2r7=!2red31; 2r8=!2red30; 2g1=!2green37;   2g2=!2green36;
    2g3=!2green35;   2g4=!2green34;   2g5=!2green33;   2g6=!2green32;
    2g7=!2green31;   2g8=!2green30;   end if;
if count.q[]==3 then    row1=gnd; row2=gnd; row3=gnd; row4=vcc; row5=gnd;
    row6=gnd; row7=gnd; row8=gnd; 1r1=!1red47; 1r2=!1red46; 1r3=!1red45;
    1r4=!1red44; 1r5=!1red43; 1r6=!1red42; 1r7=!1red41; 1r8=!1red40;
    1g1=!1green47;   1g2=!1green46;   1g3=!1green45;   1g4=!1green44;
    1g5=!1green43;   1g6=!1green42;   1g7=!1green41;   1g8=!1green40;
    2r1=!2red47; 2r2=!2red46; 2r3=!2red45; 2r4=!2red44; 2r5=!2red43;
    2r6=!2red42; 2r7=!2red41; 2r8=!2red40; 2g1=!2green47;   2g2=!2green46;
    2g3=!2green45;   2g4=!2green44;   2g5=!2green43;   2g6=!2green42;
    2g7=!2green41;   2g8=!2green40;   end if;
if count.q[]==4 then    row1=gnd; row2=gnd; row3=gnd; row4=gnd; row5=vcc;
    row6=gnd; row7=gnd; row8=gnd; 1r1=!1red57; 1r2=!1red56; 1r3=!1red55;
```

```
    1r4=!1red54;  1r5=!1red53;  1r6=!1red52;  1r7=!1red51;  1r8=!1red50;
    1g1=!1green57;   1g2=!1green56;   1g3=!1green55;   1g4=!1green54;
    1g5=!1green53;   1g6=!1green52;   1g7=!1green51;   1g8=!1green50;
    2r1=!2red57;  2r2=!2red56;  2r3=!2red55;  2r4=!2red54;  2r5=!2red53;
    2r6=!2red52;  2r7=!2red51;  2r8=!2red50;  2g1=!2green57;   2g2=!2green56;
    2g3=!2green55;   2g4=!2green54;   2g5=!2green53;   2g6=!2green52;
    2g7=!2green51;   2g8=!2green50;   end if;
if count.q[]==5 then    row1=gnd; row2=gnd; row3=gnd; row4=gnd; row5=gnd;
    row6=vcc; row7=gnd; row8=gnd; 1r1=!1red67;  1r2=!1red66;  1r3=!1red65;
    1r4=!1red64;  1r5=!1red63;  1r6=!1red62;  1r7=!1red61;  1r8=!1red60;
    1g1=!1green67;   1g2=!1green66;   1g3=!1green65;   1g4=!1green64;
    1g5=!1green63;   1g6=!1green62;   1g7=!1green61;   1g8=!1green60;
    2r1=!2red67;  2r2=!2red66;  2r3=!2red65;  2r4=!2red64;  2r5=!2red63;
    2r6=!2red62;  2r7=!2red61;  2r8=!2red60;  2g1=!2green67;   2g2=!2green66;
    2g3=!2green65;   2g4=!2green64;   2g5=!2green63;   2g6=!2green62;
    2g7=!2green61;   2g8=!2green60;   end if;
if count.q[]==6 then    row1=gnd; row2=gnd; row3=gnd; row4=gnd; row5=gnd;
    row6=gnd; row7=vcc; row8=gnd; 1r1=!1red77;  1r2=!1red76;  1r3=!1red75;
    1r4=!1red74;  1r5=!1red73;  1r6=!1red72;  1r7=!1red71;  1r8=!1red70;
    1g1=!1green77;   1g2=!1green76;   1g3=!1green75;   1g4=!1green74;
    1g5=!1green73;   1g6=!1green72;   1g7=!1green71;   1g8=!1green70;
    2r1=!2red77;  2r2=!2red76;  2r3=!2red75;  2r4=!2red74;  2r5=!2red73;
    2r6=!2red72;  2r7=!2red71;  2r8=!2red70;  2g1=!2green77;   2g2=!2green76;
    2g3=!2green75;   2g4=!2green74;   2g5=!2green73;   2g6=!2green72;
    2g7=!2green71;   2g8=!2green70;   end if;
if count.q[]==7 then    row1=gnd; row2=gnd; row3=gnd; row4=gnd; row5=gnd;
    row6=gnd; row7=gnd; row8=vcc; 1r1=!1red87;  1r2=!1red86;  1r3=!1red85;
    1r4=!1red84;  1r5=!1red83;  1r6=!1red82;  1r7=!1red81;  1r8=!1red80;
    1g1=!1green87;   1g2=!1green86;   1g3=!1green85;   1g4=!1green84;
    1g5=!1green83;   1g6=!1green82;   1g7=!1green81;   1g8=!1green80;
    2r1=!2red87;  2r2=!2red86;  2r3=!2red85;  2r4=!2red84;  2r5=!2red83;
    2r6=!2red82;  2r7=!2red81;  2r8=!2red80;  2g1=!2green87;   2g2=!2green86;
    2g3=!2green85;   2g4=!2green84;   2g5=!2green83;   2g6=!2green82;
    2g7=!2green81;   2g8=!2green80;   end if;
end;
```

# 设计感想

1. AHDL 语言不同于 C 语言。最重要的区别是 AHDL 语言的所有代码是同时执行的、C 语言是依次执行的。要是代码依次执行(如音乐序列)，必须使用计数器。
2. 根据 Altera 公司在帮助文档中给出的建议，我采用了大量的 lpm_counter 组件实现计数器。有人认为 lpm_counter 占用资源太大、应该用 machine with states，我认为 lpm_counter 可以加速开发，且在提供了合适的参数(如限定 lpm_direction)后，占用资源应该可以下降。
3. 为了更高效的开发，应该合理使用组件，这有利于重用现有的代码。
4. 多用 tdf。tdf 虽然结构上并不直观，但是功能定制更容易，录入速度更快。有人认为 tdf 会比 gdf 多占资源，我并不赞同，相同功能的 tdf 应是和 gdf 等价的。