武汉大学计算机学院
本科生实验报告

基于 AT89C52 的电梯控制系统

专 业 名 称 ：计算机科学与技术

课 程 名 称 ： 嵌入式系统

指 导 教 师 ： 武小平 副教授

学 生 学 号 ： 2018302110219

学 生 姓 名 ： 何 华

二〇二一年四月

郑 重 声 明

本人呈交的实验报告，是在指导老师的指导下，独立进行实验工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本实验报告不包含他人享有著作权的内容。对本实验报告做出贡献的其他个人和集体，均已在文中以明确的方式标明。本实验报告的知识产权归属于培养单位。

本人签名：何华　　日期：2021.04.30

摘　要

本次实验我采用 AT89C52 单片机完成了八层双电梯的调度系统仿真。整个系统可以分为两部分，基于 Proteus 的仿真系统和基于有穷状态机的调度算法。仿真系统可以进一步的划分为两个部分，即基于 LCD1602 液晶显示器的状态显示电路和基于 74LS148 译码器，74LS138 编码器的电梯控制电路；基于有穷状态机的调度算法利用位运算将整数作为队列，将请求视为转移动作。达到了比较好的显示效果和比较好的调度效率、效果。

关键词：AT89C52 单片机　电梯调度　有穷状态机

目录

1 实验目的和意义

1 实验目的

通过设计一个基于单片机的电梯控制系统，增强同学们对单片机工作原理的理解，掌握嵌入式系统设计的基本步骤，学习并掌握使用 Proteus 进行单片机仿真和使用 C 语言进行单片机片成，掌握嵌入式系统开发的基本流程，激发同学们对于嵌入式系统的兴趣。

2 实验意义

本次实验通过让学生基于单片机，Proteus 设计一个电梯调度系统，具有以下意义：

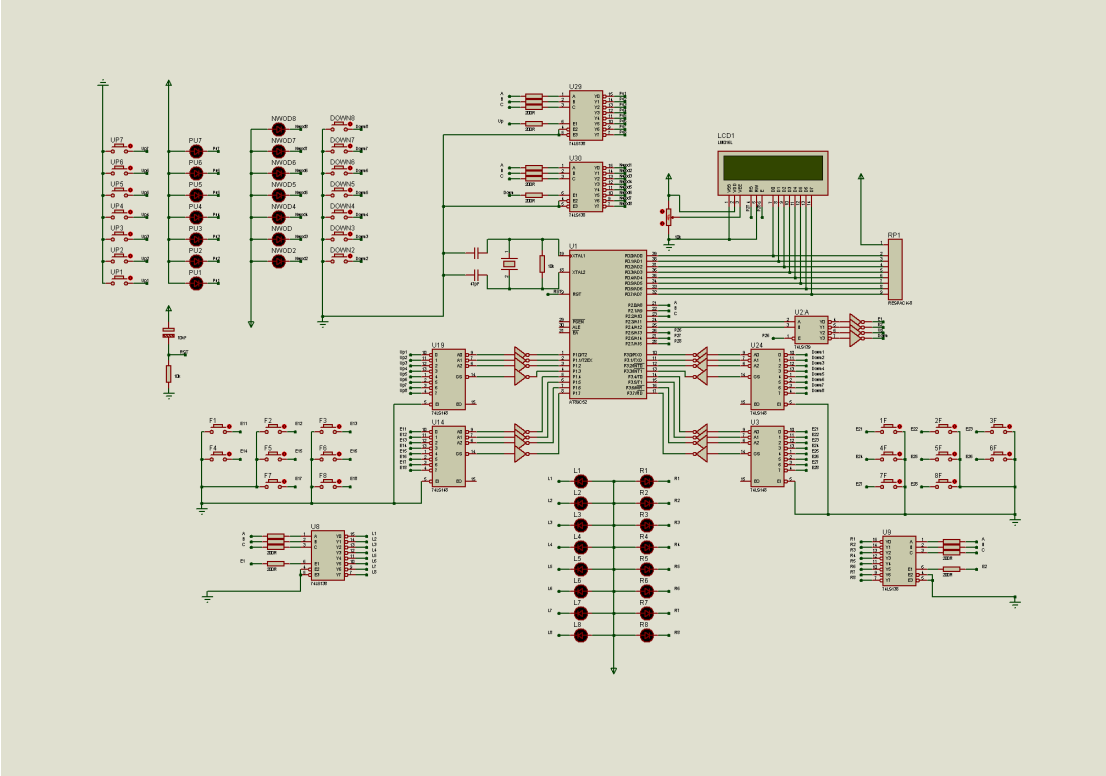
- 增强学生对于 51 单片机工作原理的理解
- 使同学们熟悉并掌握使用 Proteus，C 语言进行设计和仿真的流程
- 了解嵌入式系统开发的基本流程
- 激发同学们对于嵌入式式系统的兴趣
- 使同学们了解并掌握基本的电梯调度算法，如 SCAN，LOOK，C-LOOK，有穷状态机等算法的设计

2　实验设计

1　硬件设计原理

1.1　原理图

本次实验我基于 AT89C52 单片机完成了八层双电梯的调度系统，其中 Proteus 仿真中用的原理图如下：



整个原理图由三个部分组成，中间的 AT89C52 单片机，右上角的基于 LCD1602 液晶显示屏的状态显示电路和四个电梯控制电路。其中，每个电梯控制电路通过 74LS148 编码器将 8 个按钮编码成一个整数传送给单片机，单片机对 4 个整数值（分别代表外部向上请求，外部向下请求，左电梯内部请求和右电梯内部请求）进行处理后，将处理结果先输送给液晶显示器，然后通过 74LS138 解码器对对应结果解码，并控制对应的楼层指示灯，达到较好的控制效果。

由于整个系统比较复杂，接下来我将分状态显示电路和电梯控制电路两个部分进行硬件设计原理的说明。

1.2 局部原理图说明

1.2.1 状态显示电路

前面已经提到，我们的状态显示电路是基于 LCD1602 液晶显示器，在 Proteus 中，元器件名为 LM016L。以下是截取自 LM016L Datasheet 的管脚连接表格：

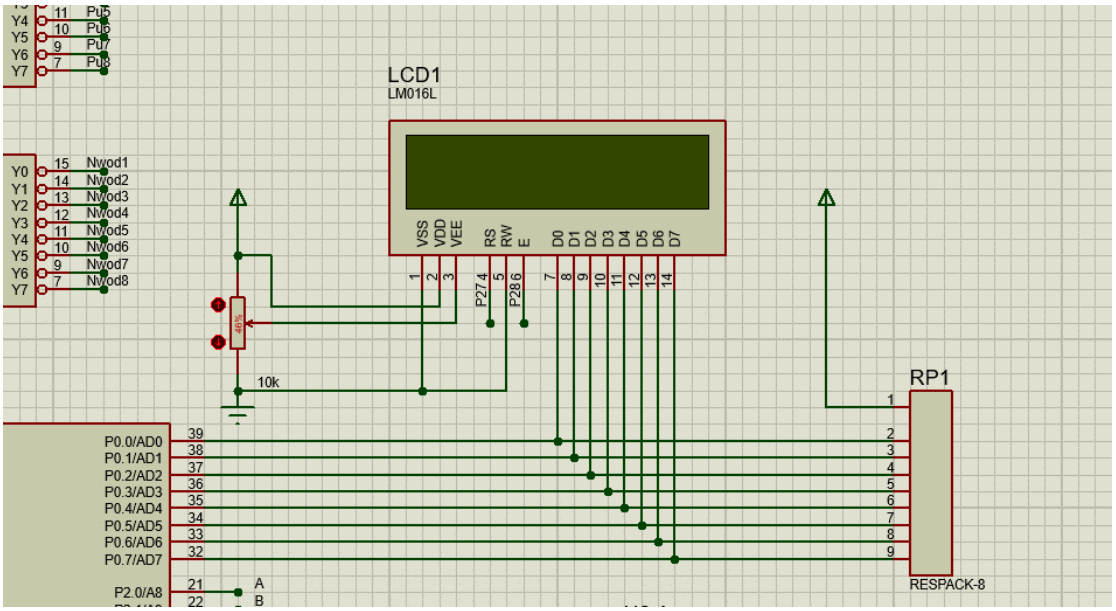
INTERNAL PIN CONNECTION

Pin No.	Symbol	Level	Function	
1	V_{SS}	–	0V	Power supply
2	V_{DD}	–	+5V	
3	V_O	–	–	
4	RS	H/L	L: Instruction code input H: Data input	
5	R/W	H/L	H: Data read (LCD module→MPU) L: Data write (LCD module←MPU)	
6	E	H, H→L	Enable signal	
7	DB0	H/L	Data bus line Note (1), (2)	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L		
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		

可以看到，我们需要将 VSS 接地，VDD 接电源，VO 主要用来控制液晶显示器的对比度，这里我们将它连接到一个滑动变阻器；RS 用来控制写命令还是写数据，我们将其连接到单片机的 P2_7 端口；RW 用来控制从 LM016L 读数据或者写数据，容易想到，我们的电梯状态显示电路不需要从 LM016L 读数据，这里为了节约管脚将其直接接地；E 为实能端口，将其接到单片机的 P2_8 端口。接下来的 DB0-DB7 端口主要用来向 LM016L 写入数据，因此我们将其连接到单片机的 P0 端口。

此外，还需要注意的是：当 P0 端口不扩展时，可以做一般的 I/O 端口使用，但是其内部没有上拉电阻，因此，当我们用 P0 端口作为输出的时候，我们需要手动给 P0 的每个端口配备上拉电阻。

根据上述分析，我们很容易得到如下的电梯系统状态显示电路：



1.2.2 电梯控制电路

1.2.2.1 按钮检测电路

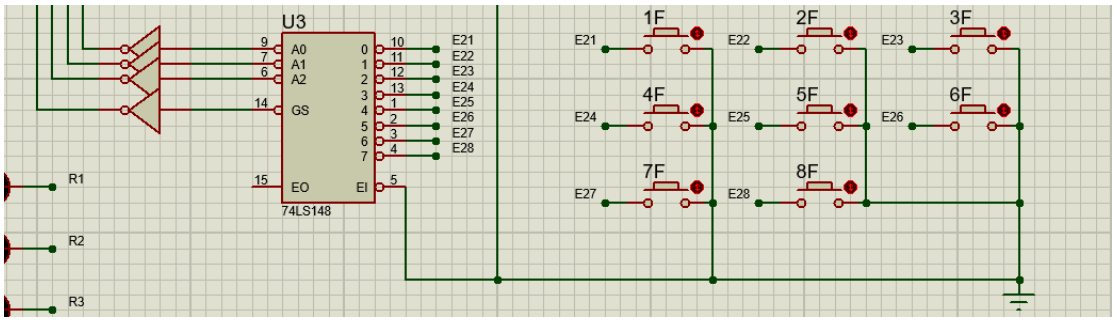
在摘要部分我们说到过，我们的仿真中有四个电梯控制电路，即外部向上请求电路，外部向下请求电路，左电梯内部请求电路，右电梯内部请求电路。其中的每个电梯控制电路由 8 个按钮，8 个 LED 灯组成。当按下对应的按钮时，对应的 LED 亮；当电梯满足对应的请求时，对应的 LED 灯灭。

可以想到，如果我们直接将这 32 个元器件连接到单片机，IO 管脚是远远不够的。因此我们得想办法扩展 IO 管脚，很容易可以想到，我们可以将 8 个按钮进行编码，并在单片机中进行解码。

因此我们需要使用 8-3 编码器 74LS148，以下是其真值表：

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

观察 74LS148 的真值表可以看出，如果我们将 EI 位使能，那么如果每一个按钮的按下可以对应一个低电平，那么输出的 A2A1A0 就可以唯一对应一个状态。而如果我们想将 A2A1A0 唯一的映射到 1-8 中的一个位置，我们需要将 A2A1A0 取反并加一。还有一点值得注意的是：观察 A2A1A0 我们可以发现，当没有按钮被按下，或者第 1 个按钮被按下（也就是对应第 0 位输入为低）的时候，A2A1A0 的输出是一样的，也就没有办法区分，我们得想办法区分这两个状态；再观察 74LS148 的输出可以发现，GS 端口在有按钮被按下和没按钮被按下时其电平不一样，因此我们可以使用 GS 端口来判断是否有按钮被按下。也就可以得到如下的设计：



1.2.2.2 指示灯解码电路

和按钮检测电路类似的思路，既然我们可以使用 74LS148 编码器对按钮的按下状态进行编码，那么我们也应该可以使用 74LS138 译码器对单片机的控制信号进行解码，以此来进一步节省 IO 管脚的使用。以下是 74LS138 译码器的真值表：

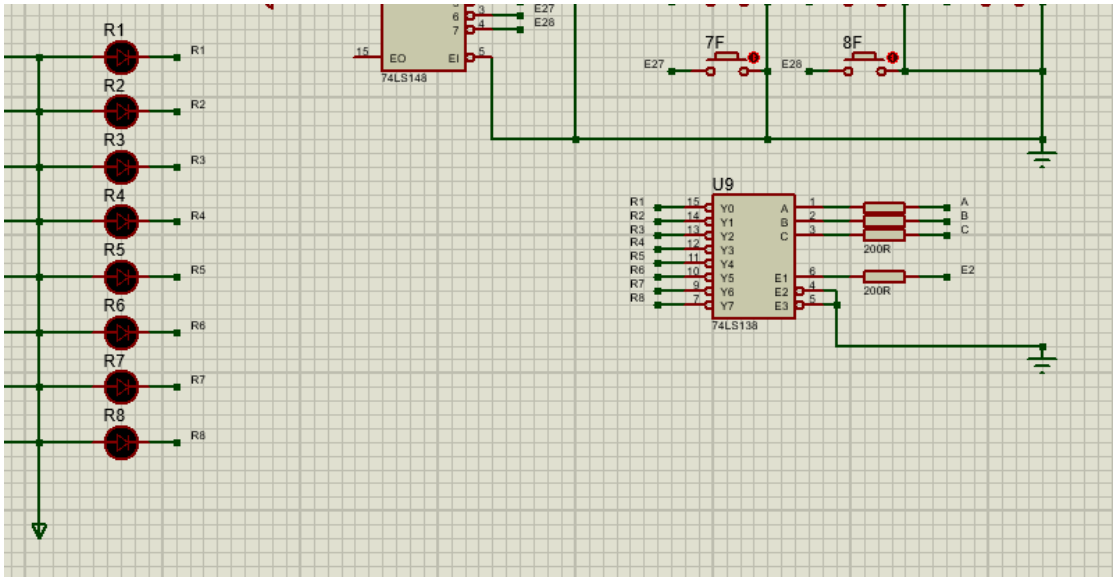
Inputs					Outputs							
Enable		Select										
G1	G2 (Note 1)	C	B	A	YO	Y1	Y2	Y3	Y4	Y5	Y6	Y7
X	H	X	X	X	H	H	H	H	H	H	H	H
L	X	X	X	X	H	H	H	H	H	H	H	H
H	L	L	L	L	L	H	H	H	H	H	H	H
H	L	L	L	H	H	L	H	H	H	H	H	H
H	L	L	H	L	H	H	L	H	H	H	H	H
H	L	L	H	H	H	H	H	L	H	H	H	H
H	L	H	L	L	H	H	H	H	L	H	H	H
H	L	H	L	H	H	H	H	H	H	L	H	H
H	L	H	H	L	H	H	H	H	H	H	L	H
H	L	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Level
L = LOW Level
X = Don't Care

Note 1: G2 = G2A + G2B

可以看到，如果我们置 G1 端口为 1，G2 端口（G2A+G2B）为 0，那么对于选

通端 CBA，我们可以将一个整数唯一的映射为一个输出的状态，在共阳极的 8 个 LED 灯间点亮一个 LED 灯。但也要注意到，74LS138 的八个输出端口里，一次只能有一个端口输出为低电平，我们需要想办法表示多个按钮同时按下的情况，也就是多个 LED 灯同时点亮。注意我们这里不能使用锁存器来保持 LED 的状态（锁存器只能对输入的 8 个端口进行锁存，而输入的 8 个端口同时还是只能有一个低电平）。由于没能想到更好的办法来保持多个灯同时亮的情况，我在这里使用代码扫描的方式（循环）来维持多个灯“同时点亮”。以下是指示灯解码电路的原理图：



我们将 CBA 分别连接到 P2_2、P2_1、P2_0 三个端口，将 E2、E3 端口直接接地。通过这种方式实现对 P2_2、P2_1、P2_0 三个端口控制信号的译码，使某个管脚输出为低电平，点亮对应的 LED。

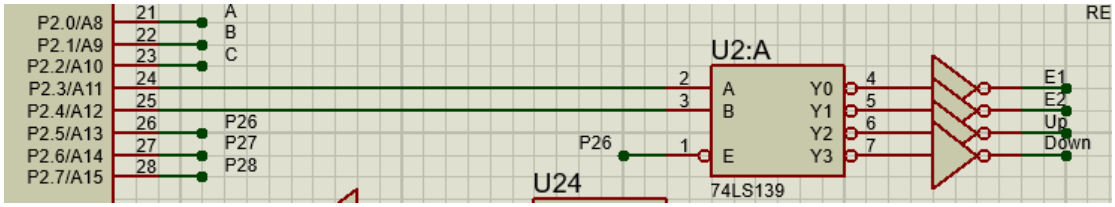
那 E1 端口起什么作用呢？我们首先来计算一下已经使用的 IO 端口数目，LCD1602 液晶显示屏使用了 10 个管脚，74LS148 编码器每一个需要使用 4 个管脚（包含 GS 控制管脚），一共使用了 26 个管脚，现在只剩 6 个管脚，其中这 6 个管脚至少有三个需要用作 CBA 控制信号管脚。显然，CBA 控制信号每一次只能作用于一个电梯控制电路。我们需要在四个电梯控制电路中选择一个进行使能。很自然的想到我们需要使用 74LS139 译码器实现一个使能电路。我们将在下一部分说明这个二四译码器使能电路。

1.2.2.3 译码器使能电路

既然我们要使用到二四译码器进行控制电路的使能，不妨我们先来看看二四译码器的真值表：

Inputs			Outputs			
Enable	Select					
G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

由真值表很容易想到，我们可以通过 G 端口来控制是否使能二四译码器，通过 BA 选通端口来选择一个电梯控制电路并点亮对应电路的 LED 灯。考虑到指示灯解码电路中 74LS138 高电平使能，74LS139 只能输出唯一的一位低电平，我们还需要加四个反向门。因此，可以得到如下的译码器使能电路：



以上的电梯控制电路属于四个电梯控制电路中的一个，右边电梯内部请求电路。其余的三个电梯控制电路内容基本一致。不同的是两个外部请求电路（向上和向下的）只有 7 个按钮和 7 个 LED 灯，因为外部向上请求电路最多只存在于 1-7 楼；同理，外部向下请求电路最多只存在于 2-8 楼。

2 软件功能设计

对应于硬件设计原理，我们将分为状态显示功能设计，电梯控制功能设计，电梯调度算法设计三个部分进行说明。

2.1 数据结构设计

本次实验中，我抽象出了电梯这个数据结构，每个电梯拥有电梯序号，电梯当前楼层，电梯当前状态三个成员。其对应数据结构如下：

```
typedef struct Elevator {
    uint index;
    uint floor;
    uint status;
} Elev;
```

2.2 状态显示功能设计

状态显示电路主要是基于 LCD1602 液晶显示器进行状态显示，这也就要求我们要写液晶显示器的驱动程序。为此，我参考江科大自化协的 B 站 51 单片机课程将 LCD1602 的驱动封装成了一个模块供 main.c 文件调用。其提供的函数如下：

```
// 初始化液晶显示器
void LCD_Init();

// 设置输出游标位置
void LCD_SetCursor(unsigned char Line, unsigned char Column);

// 向 LCD1602 写数据
void LCD_WriteData(unsigned char Data);

// 向 LCD1602 写命令
void LCD_ShowChar(unsigned char Line, unsigned char Column, char Char);

// 在 LCD1602 指定行和列显示字符串
void LCD_ShowString(unsigned char Line, unsigned char Column, char *String);

// 在 LCD 指定行和列显示数字
void LCD_ShowNum(unsigned char Line, unsigned char Column, unsigned int Number, unsign
ed char Length);

// 在 LCD 指定行和列显示有符号数字
void LCD_ShowSignedNum(unsigned char Line, unsigned char Column, int Number, unsigned
char Length);

// 在 LCD 指定行和列显示十六进制数字
void LCD_ShowHexNum(unsigned char Line, unsigned char Column, unsigned int Number, uns
igned char Length);

// 在 LCD 指定行和列显示二进制数字
void LCD_ShowBinNum(unsigned char Line, unsigned char Column, unsigned int Number, uns
igned char Length);
```

函数具体作用见注释，这里不赘述。值得一提的是，后面的一些 LCD_Show 开头的函数极大程度上的帮助了我调试代码，提高了编码效率。

基于以上这些接口和电梯的数据结构，很容易写出显示两个电梯状态的子函数：

```
void showElevStatus() {
    // 显示左电梯状态和楼层
    LCD_SetCursor(1, 3);
```

```
    LCD_WriteData(leftElev.status);      // 显示自定义字符位置
    LCD_ShowNum(1, 4, leftElev.floor, 2);

    LCD_SetCursor(1, 12);
    LCD_WriteData(rightElev.status);
    LCD_ShowNum(1, 13, rightElev.floor, 2);
}
```

2.3 电梯控制功能设计

对应于硬件设计原理的电梯控制电路部分，这一部分我也将分解为按钮检测功能设计，指示灯解码功能设计，译码器使能功能设计。

2.3.1 按钮检测功能设计

硬件设计原理部分提到，在设计按钮检测功能的时候，我们需要使用 74LS148 的 GS 端口来检测是否有按钮按下，并且记录对应的状态。因此，存在一个问题，我们使用什么样的数据结构来存储这些状态，第一个想到的肯定是数组，但是再深入想想，四个长度为 8 的数组，即使其类型为 unsigned char，其占用空间都很大；更重要的是，当我们使用有穷状态机做电梯调度的时候，需要进行大量的 if-else 判断，其中每个条件如果都需要花费很长的时间在访存上，那么整个电梯调度系统的效率是极低的。综合考虑这两点，在 17 级一个学长以及我室友的提示下，我选用了一个 unsigned char 来存储按钮按下的状态，选用了位运算来进行快速的置标志位，清空标志位操作。

因此，对于右边电梯的内部请求电路，我们可以写出如下的按钮检测代码：

```
if (P3_7) {
        er |= (1 << ((P3_6 * 4) | (P3_5 * 2) | P3_4));
}
```

其中使用 P3_7 连接到对应的译码器的 GS 端口，P3_6、P3_5、P3_4 分别对应于 74LS148 的 A2A1A0 的非。er 变量表示右边电梯的内部请求状态数组，通过或运算来将对应按钮位置的标志置 1。而多个按钮按下，对应于将多位置 1。

2.3.2 指示灯解码功能设计

同样，对应于硬件电路的设计，在指示灯解码的代码部分，我们需要逐位的扫描状态 unsigned char 的每一位，如果某一位为 1，那么代表对应的按钮被按下，那么通过译码器使能功能，应该点亮对应控制电路的指示灯。其过程大体上可以

用如下伪码表示：

```
func showIndicators(){
      for(i in 0..8){
         PortA <<= i & 0x01;
         PortB <<= i & 0x02;
         PortC <<= i & 0x04;

         // 以下的el, er, up, down分别表示
         // 左边电梯内部请求，右边电梯内部请求
         // 外部向上请求，内部向上请求状态集
         two2FourDecode(
             el[i]==1,
             er[i]==1,
             up[i]==1,
             down[i]==1
         );}
}
```

2.3.3 译码器使能功能设计

整个译码器使能功能，对应于一个二四译码器。因此，只要对应的状态标志为真，那么我们将选通对应的 74LS138 使能端口 E1 即可。注意，因为我们的多个电梯控制电路指示灯可以同时亮，因此，我们的四个 if 应该是并列的状态。并且，在此子函数的结尾，我们应该将译码器使能端口恢复。防止干扰后续迭代过程。其伪代码可以表示为如下：

```
func two2FourDecode(uchar flagEl, uchar flagEr, uchar flagUp, uchar flagDown){
    if(flagEl){
        使能二四译码器;
        点亮左边内部请求指示灯;
    }
    if(flagEr){
        使能二四译码器;
        点亮右边内部请求指示灯;
    }
    if(flagUp){
        使能二四译码器;
        点亮外部向上请求指示灯;
    }
    if(flagDown){
        使能二四译码器;
        点亮外部向下请求指示灯;
    }
    将二四译码器使能端清 0;
```

```
}
```

2.4 电梯调度算法设计

在查阅一些资料以及文献后，我得知：为了尽可能地提高电梯调度的效益，每个电梯的调度需要满足如下的条件：

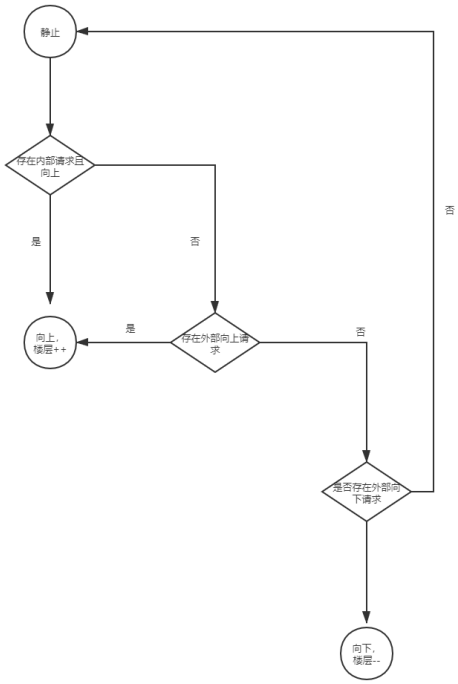
- **处于上升状态的电梯只处理向上的请求，出于向下状态的电梯只处理向下的请求**
- **当有多个向上请求的时候，先下降到最底层再响应向上的请求，当有多个向下的请求的时候，向上升到最高层再响应向下的请求**
- **多梯调度的时候，处于更高楼层的处理比自己更高楼层的所有外部向上请求和自己的内部请求，处于更低楼层的处理比自己更低楼层的所有外部向下请求和自己的内部请求**

由数据结构设计部分可以知道，每个电梯有三种状态，即上升、下降以及静止。那么两个电梯就一共有如下的九种状态：

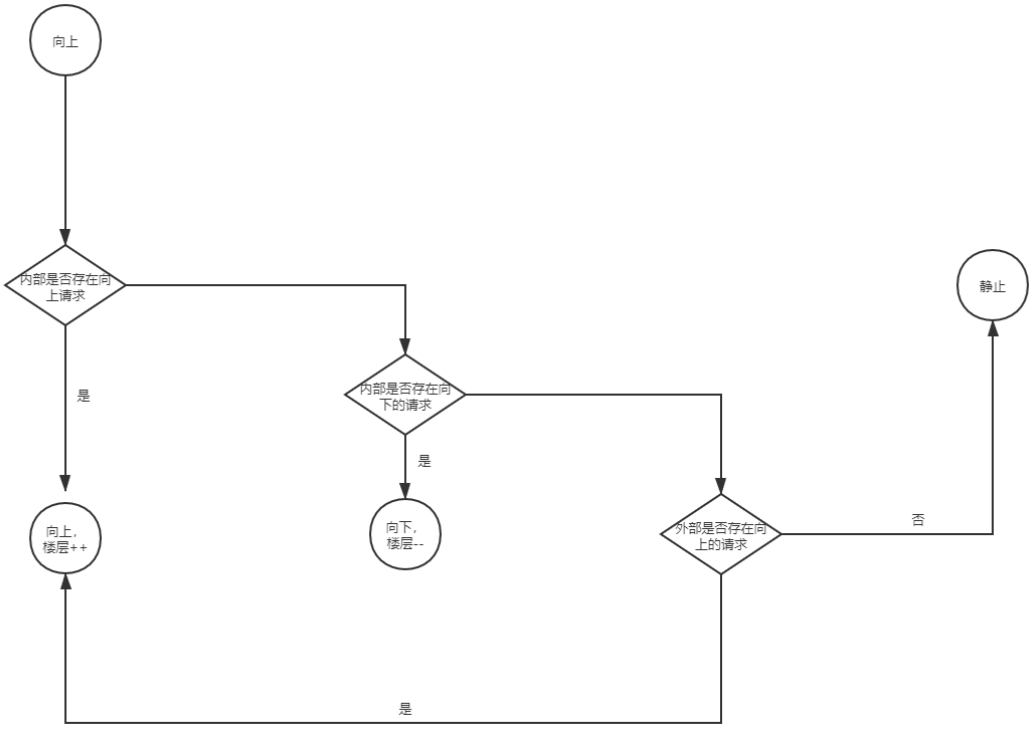
	静止	上升	下降
静止	静止，静止	静止，上升	静止，下降
上升	上升，静止	上升，上升	上升，下降
下降	下降，静止	下降，上升	下降，下降

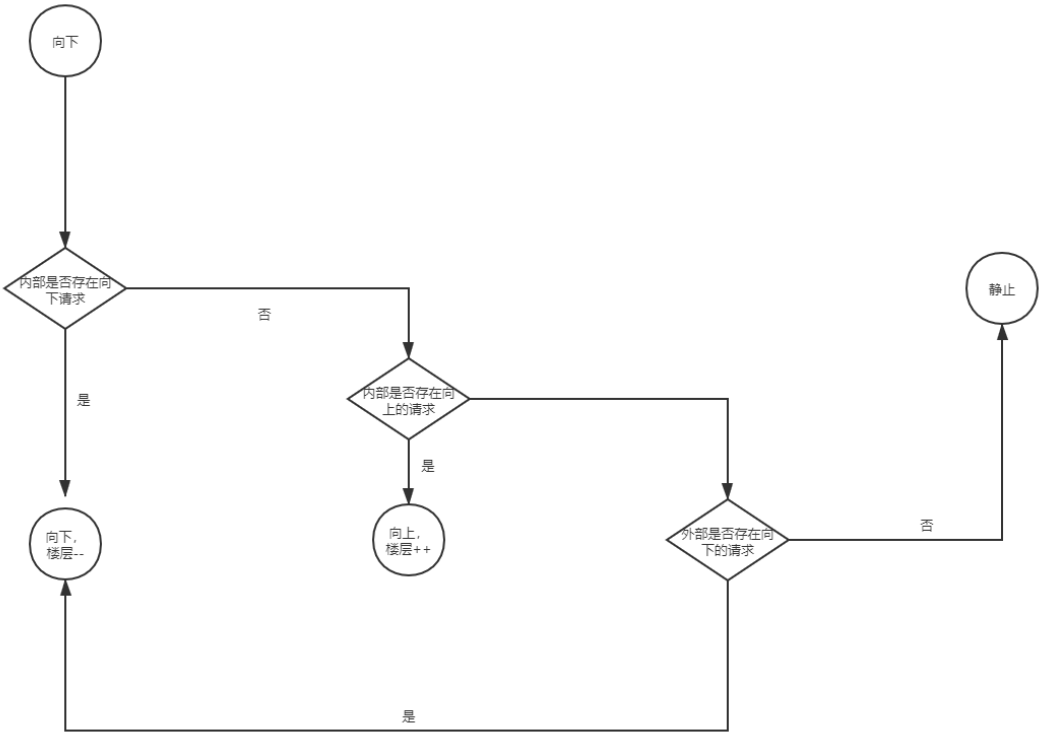
同时，我们知道两个电梯共用四个状态变量，即 up(外部电梯向上请求), down(外部电梯向下请求), el(左边电梯内部请求)和 er(右边电梯内部请求)。但是每个电梯只用响应三个状态变量，对于左边的电梯，是 el, up, down; 对于右边的电梯是，er, up, down。如果我们让每个电梯优先响应自己的内部的请求，那么我们就可以根据这四个状态变量做出一个有穷状态机。

具体的来说，对于两个电梯，如果两个电梯都处于静止的状态，那么两个电梯都做如下图的状态转换：



如果一个电梯处于静止状态，一个电梯处于运动状态，处于静止状态的电梯做如上图的转换，处于运动状态的电梯做如下图的状态转换：





注：以上的每个图的初始状态都是左上角第一个圆圈

对于两个电梯都运动的情况，我们可以在如上图的两个状态转换图中找到对应的状态转换。并做出相应的响应。

由于代码在 word 中显示效果并不太好，这里就不粘贴各个分支的代码。如果老师或者助教又兴趣查看对应部分的代码，可以访问我的 Github 仓库：

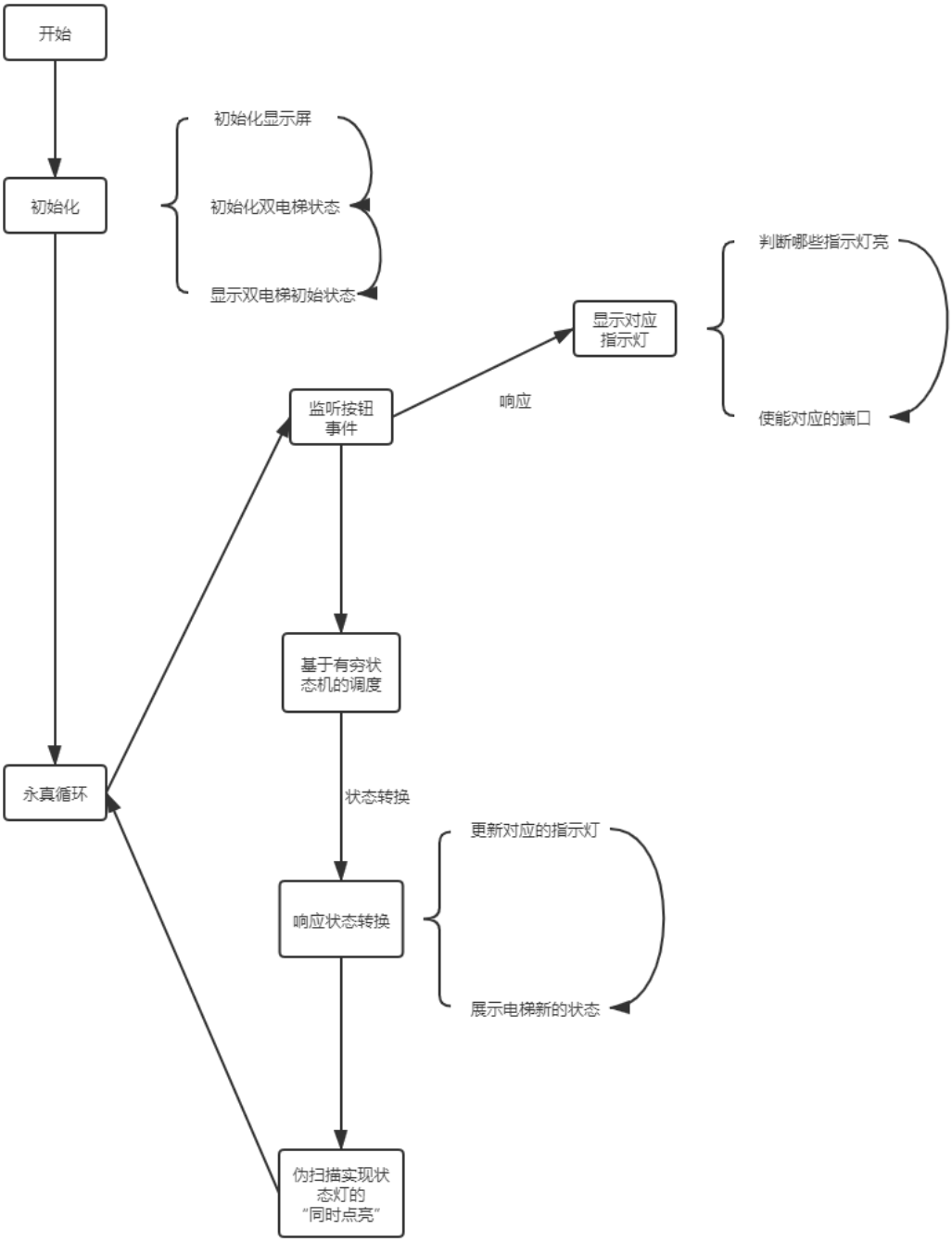
https://github.com/HwaHe/Elevator-Simulation

或者，访问在线 vscode 环境查看高亮状态的代码：

https://github1s.com/HwaHe/Elevator-Simulation/blob/HEAD/keil/main.c

3　实现流程

由于在第二部分实验设计中从各个小的角度对实现过程进行了比较充实的说明，在这部分我们只展示代码的整体框架流程，以便从一个更广的角度展示我的电梯调度系统：



4　遇到的问题与尚存在的问题

1　遇到的问题

在本次设计解决方案的过程中我也遇到了许多的问题，以下选取部分进行记录以及说明：

1. Proteus 频繁闪退，所做的工作无法保存，基本无法仿真。原因在于破解版软件的破解程序在 ProgramData 目录下的 Proteus 目录并没有复制进去 Proteus 仿真需要的 Model 和 Bin 目录。
2. LED 闪烁速度过快，无法常亮。由于要模拟多层按钮被按下的情况，但是 74LS138 译码器一次只能点亮一个 LED 灯，我开始计划使用循环扫描的方式，达到一种“伪同时亮”，但 LED 一直闪烁，演示效果并不理想。后经室友提醒，可以修改 LED 灯的“Minimum on time to light”属性来达到 LED 灯更长的点亮时间，以此对 LED 灯常量进行模拟。
3. 考虑过很多电梯调度算法，企图设计一个最优的电梯调度系统，未果。在查阅很多资料后发现，多电梯调度问题可以归结于一个基于时间的旅行商问题，属于 NP-Hard 问题，不存在最优解，包括 SCAN 算法，LOOK 算法都只是一个可接受解。传统算法更多的是将多梯调度考虑成一个状态机，每一个请求可能引起电梯的状态发生改变，问题在于我们怎么合理的选择出可能引起电梯状态发生改变的请求，也因此导出了我的基于有限状态机的电梯调度算法。

2　尚存在的问题

1. 基于状态机的电梯调度算法响应速度过慢，因为要进行大量的条件判断。
2. 当两个电梯在不同的高度静止时，如果中间有外部请求，则两个电梯都会响应，显然这是不合理的。
3. 当有外部的请求时，如果两个电梯都静止且无内部请求，两个电梯都会朝对应方向移动，很容易想到，这对距离请求更远的电梯是不必要的，也是不合理的。

5　总结

这次嵌入式课程设计可以算得上我的第一次嵌入式编程体验。从如何对抗软件的 crash 到如何合理的利用 IO 管脚，设计方便调试，方便使用的原理图到如何选择一个更好用的编码软件，如何实现一个仿真效果良好的调度算法，这其中遇到了许许多多困难以及崩溃的瞬间，尤其是晚上一点画好图保存不了的时候…但是合理的减少了 IO 管脚的使用的时候，找到了合适的数据结构的时候，调度算法仿真效果还不错的时候，还是觉得嵌入式系统设计挺有趣的，也激发了我对嵌入式系统编程的兴趣，学到了很多嵌入式系统编程，仿真的知识。

教师评语评分

评语：_____

_____

_____

_____

_____

_____

_____

_____

评分：_____

评阅人：

年　　月　　日