一种全自动洗衣机控制系统开发

1.学习目标

1. 能根据控制系统任务要求完成直流电机的正反停转控制和 PWM 调速控制；
2. 能按照功能要求执行顺序制定程序设计分步实施方案；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

单片机是一种在半导体硅片上集成了中央处理器（CPU）、存储器（RAM、ROM）、输入/输出接口（I/O 接口）、定时器/计数器等功能的单片微型计算机。自问世以来，单片机以其功能全、体积小、价格低的优势广泛应用于智能仪表、机电设备、工业控制、家用电器等多个领域。本书以美国 ATMEL 公司的 AT89C52 单片机为例介绍单片机在各个领域的多项应用设计案例。

直流电机可以将电信号转换为角位移或线位移等机械信号，是单片机应用中最常见的被控器件。本章提出一例“设计一种全自动洗衣机控制系统”的课题需求，学习直流电机的旋转调速控制，并掌握单片机顺序结构控制系统的程序设计方法。

任务题目：设计一种全自动洗衣机控制系统

功能要求：该系统能够按照设定的功能顺序控制直流电机的转停、方向、速度，以模拟全自动洗衣机的工作过程。其中洗衣机的工作过程可以分为洗涤、漂洗、脱水三项，洗涤过程采用“3 秒洗涤工作循环”方式进行 3 次，漂洗过程采用“2 秒漂洗工作循环”方式进行 3 次，脱水过程采用“脱水工作”方式。具体控制要求如下：

1.洗涤功能。设定“n 秒洗涤工作循环”方式为电机正转 n 秒，停止 0.5 秒，反转 n 秒，停止 0.5 秒，以 n 秒时长的电机正反转构成 1 个工作循环，称为 n 秒洗涤工作循环。

系统开机首先进入洗涤过程，采用“3 秒洗涤工作循环”方式进行 3 次，即电机正转 3 秒，停止 0.5 秒，反转 3 秒，停止 0.5 秒，如此循环 3 次，使衣物在水中翻转摩擦。

2.漂洗功能。设定“n 秒漂洗工作循环”方式为电机以 50%占空比的速度正转 n 秒，停止 0.5 秒，以 50%占空比的速度反转 n 秒，停止 0.5 秒，以 n 秒时长的电机慢速正反转构成 1 个工作循环，称为 n 秒漂洗工作循环。

在完成洗涤过程后，系统自动进入漂洗过程，采用“2 秒漂洗工作循环”方式进行 3 次，即电机以 50%占空比的速度正转 2 秒，停止 0.5 秒，以 50%占空比的速度反转 2 秒，停止 0.5 秒，如此循环 3 次，使衣物在水中散开分离。

3.脱水功能。设定“脱水工作”方式为电机全速正转 5 秒后关闭，保持惯性直至停止。

在完成漂洗过程后，系统自动进入脱水过程，采用“脱水工作”方式，即电机全速正转 5 秒后关闭，保持惯性直至停止，使衣物高速甩干。至此系统控制洗衣机完成一次全自动洗衣流程。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输出模块

单片机输出模块包括直流电机控制模块，主要用于洗衣机洗涤、漂洗、脱水过程中的旋转调速控制。该功能由单片机输出信号控制电机驱动，对旋转角度的控制要求不高，可以选择直流电机及其驱动模块的控制方案。同时该模块的选型应根据实际控制装置的要求进行，本例不作展开。

单片机通过两个引脚输出控制信号至驱动模块，从而实现电机的正反停转控制和 PWM 调速控制。直流电机接口电路如图 1.1 所示，在开发板上占用 P2.6/P2.7 引脚。设定当 P2.6=1、P2.7=0 时为电机正转，当 P2.6=0、P2.7=1 时为电机反转，

当 P2.6=P2.7=0 或 P2.6=P2.7=1 时为电机停转。另外，可以通过施加一个 PWM 脉冲波控制高电平占比来进行电机调速控制，其中 PWM 信号的占空比即为对应的电机速度。

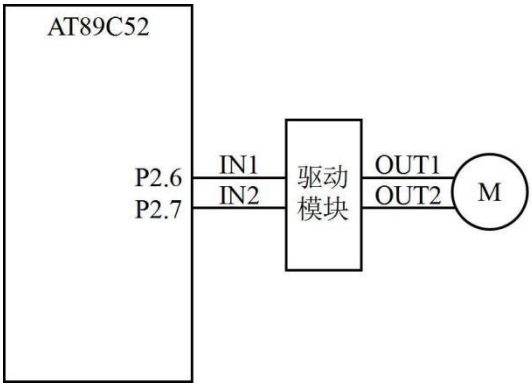


图 1.1 直流电机接口电路图

3.2 单片机内部配置

系统端口及硬件配置如表 1.1 所示。

表 1.1 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P2.6/P2.7	直流电机

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“洗涤”、“漂洗”、“脱水”三个模块，系统上电后应按照功能顺序依次执行这三个模块。因此，系统软件架构如图 1.2 所示。

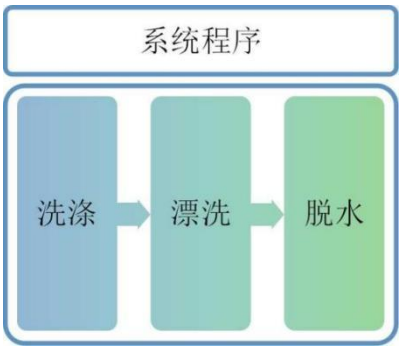


图 1.2 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由三个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 1.2 所示。

表 1.2 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	洗涤控制模块设计	按给定参数的电机正反停转控制
Step 2	漂洗控制模块设计	按给定参数的电机 PWM 调速控制
Step 3	脱水控制模块设计	

4.2.1 step1：洗涤控制模块设计

本步骤主要完成的具体任务是采用“3 秒洗涤工作循环”方式进行 3 次，即电机正转 3 秒，停止 0.5 秒，反转 3 秒，停止 0.5 秒，如此循环 3 次，程序设计流程图如图 1.3 所示。需要注意的是，本例设定了“n 秒洗涤工作循环”方式，将洗涤过程的工作时间和循环次数设为变量能够提高系统控制的灵活程度，另外，调用 void 型函数也能够使系统程序更为结构化和模块化。

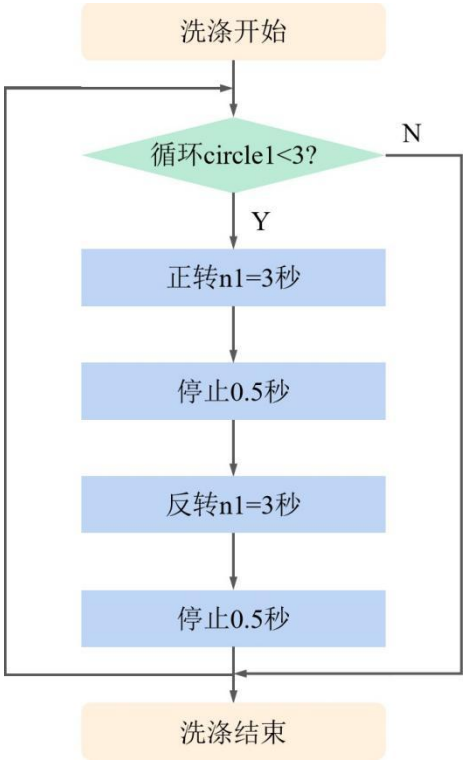


图 1.3 洗涤模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现直流电机的正反停转控制，然后在此基础上实现单次“3 秒洗涤工作循环”控制，最后调用单次洗涤函数实现 3 次循环自动洗涤控制。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，定义 motor_a/motor_b 为电机端口 P2.6/P2.7，根据电机转向控制的工作原理分别建立电机的正转、反转、停止函数，并声明程序中使用的所有函数。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#define uchar unsigned char
sbit motor_a=P2^6;                    //电机端口 P2.6 定义
sbit motor_b=P2^7;                    //电机端口 P2.7 定义
void fwd();                           //电机正转函数声明
void back();                          //电机反转函数声明
void stop();                          //电机停止函数声明

void fwd()                            //电机正转函数
{
    motor_a=1;
    motor_b=0;
}
void back()                           //电机反转函数
```

```
{
    motor_a=0;
    motor_b=1;
}
void stop()                          //电机停止函数
{
    motor_a=1;
    motor_b=1;
}
```

（2）调用正转、反转、停止函数建立“3 秒洗涤工作循环”洗涤函数，其中电机正转 3 秒，停止 0.5 秒，反转 3 秒，停止 0.5 秒。此外，将洗涤工作时间设为变量 n1，通过使用 if 语句循环延时 1000ms 函数调整。所涉及的延时函数可以在 STC 软件的软件延时计算器中得到，设置系统频率为 11.0592MHz，选择指令集为 STC-Y1，如图 1.4 所示。

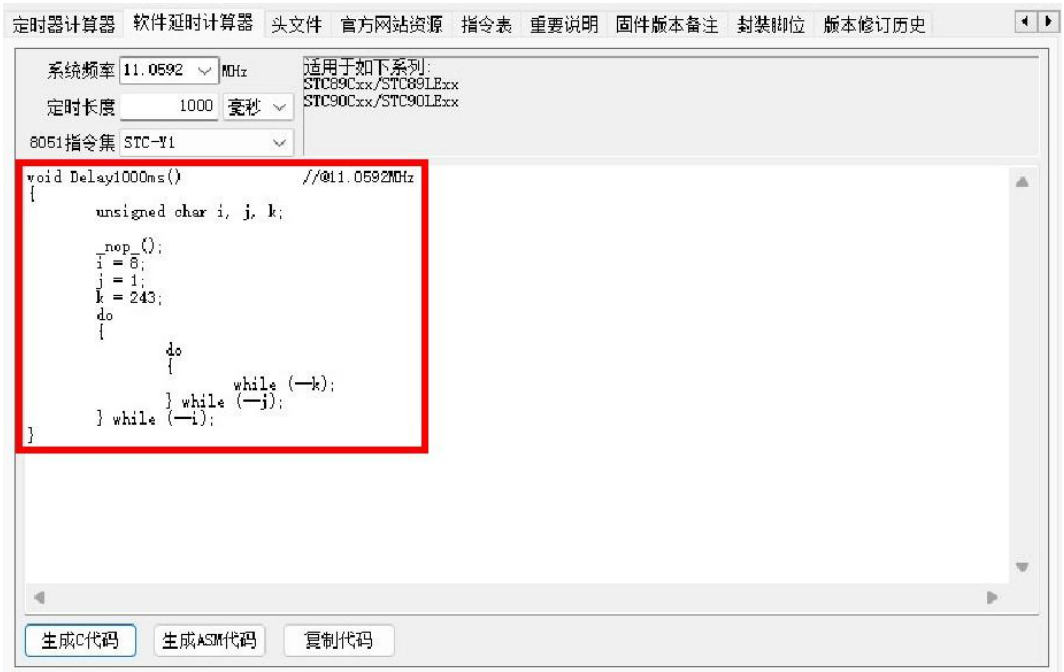


图 1.4 STC 软件延时计算器界面

相应程序如下：

```
...
void Delay500ms();                   //延时 500ms 函数声明
void Delay1000ms();                  //延时 1000ms 函数声明
void wash(uchar n1);                 //洗涤函数声明

void wash(uchar n1)                  //洗涤函数
{
    uchar k1;
```

```
    fwd();                          //电机正转
    for(k1=0;k1<n1;k1++)            //延时 n1*1000ms
    {
        Delay1000ms();
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
    back();                         //电机反转
    for(k1=0;k1<n1;k1++)            //延时 n1*1000ms
    {
        Delay1000ms();
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
}

void Delay500ms()                   //延时 500ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 4;
    j = 129;
    k = 119;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}

void Delay1000ms()                  //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
```

```
        } while (--j);
    } while (--i);
}
```

（3）调用“3 秒洗涤工作循环”洗涤函数建立 3 次循环自动洗涤函数，将洗涤循环次数设为变量 circle1，通过 if 语句循环洗涤函数调整。同时，在主函数中调用自动洗涤函数，根据功能要求设定洗涤工作时间 n1 为 3 秒，洗涤循环次数 circle1 为 3 次。相应程序如下：

```
...
void multy_wash(uchar n1,uchar circle1);            //自动洗涤函数声明

void main()                                         //主函数
{
    multy_wash(3,3);                                //“3 秒洗涤工作循环”进行 3 次
    while(1);
}

void multy_wash(uchar n1,uchar circle1)             //自动洗涤函数
{
    uchar t1;
    for(t1=0;t1<circle1;t1++)                       //循环 circle1 次
    {
        wash(n1);                                   //单次洗涤
    }
}
```

4.2.2 step2：漂洗控制模块设计

本步骤主要完成的具体任务是采用“2 秒漂洗工作循环”方式进行 3 次，即电机以 50%占空比的速度正转 2 秒，停止 0.5 秒，以 50%占空比的速度反转 2 秒，停止 0.5 秒，如此循环 3 次，程序设计流程图如图 1.5 所示。与洗涤控制模块设计相同的是，本例设定了“n 秒漂洗工作循环”方式，将漂洗过程的工作时间和循环次数设为变量能够提高系统控制的灵活程度。与洗涤控制模块设计不同的是，漂洗过程要求电机以 50%占空比的速度工作，通过施加一个 PWM 脉冲波控制高电平占比来进行电机调速控制。具体可以设定一个工作周期，并在单个工作周期内按照所要求的占空比正/反转和停止相应的时长即可。

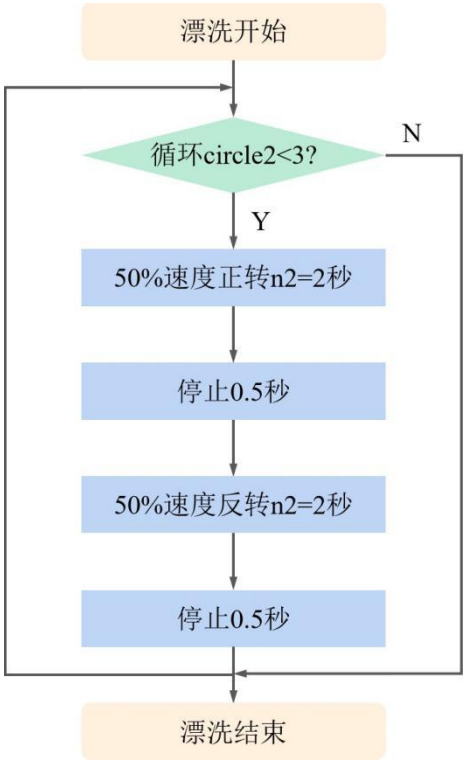


图 1.5 漂洗模块程序设计流程图

针对本步骤的设计任务，设计思路同样应考虑在直流电机的正反停转控制基础上实现单次“2 秒漂洗工作循环”控制，然后调用单次漂洗函数实现 3 次循环自动漂洗控制。

（1）调用正转、反转、停止函数建立“2 秒洗涤工作循环”洗涤函数，其中电机以 50%占空比的速度正转 2 秒，停止 0.5 秒，以 50%占空比的速度反转 2 秒，停止 0.5 秒。设定一个工作周期为 20ms，在单个工作周期内使电机正/反转 10ms，停止 10ms，如此电机正/反转时长占总时长的 50%，即可实现电机以 50%占空比的速度工作。同样将漂洗工作时间设为变量 n2，工作时间与工作周期的关系为 1 秒 50 个周期，将工作时间转化为循环工作周期的次数，通过使用 if 语句循环相应次数调整。相应程序如下：

```
...
void Delay10ms();                    //延时 10ms 函数声明
void rinse(uchar n2);                //漂洗函数声明

void rinse(uchar n2)                 //漂洗函数
{
    uchar k2;
    n2=50*n2;                        //将工作时间转化为循环次数
    for(k2=0;k2<n2;k2++)
```

```
    {
        fwd();                          //电机正转
        Delay10ms();                    //延时 10ms
        stop();                         //电机停止
        Delay10ms();                    //延时 10ms
    }
    stop();                             //电机停止
    Delay500ms();                       //延时 500ms
    for(k2=0;k2<n2;k2++)
    {
        back();                         //电机反转
        Delay10ms();                    //延时 10ms
        stop();                         //电机停止
        Delay10ms();                    //延时 10ms
    }
    stop();                             //电机停止
    Delay500ms();                       //延时 500ms
}

void Delay10ms()                        //延时 10ms 函数
{
    unsigned char i, j;
    i = 18;
    j = 235;
    do
    {
        while (--j);
    } while (--i);
}
```

（2）调用“2 秒漂洗工作循环”漂洗函数建立 3 次循环自动漂洗函数，同样将漂洗循环次数设为变量 circle2，通过 if 语句循环漂洗函数调整。同时，在主函数中调用自动漂洗函数，根据功能要求设定漂洗工作时间 n2 为 2 秒，漂洗循环次数 circle2 为 3 次。相应程序如下：

```
...
void multy_rinse(uchar n2,uchar circle2);           //自动漂洗函数声明

void main()                             //主函数
{
    multy_wash(3,3);                    //“3 秒洗涤工作循环”进行 3 次
    multy_rinse(2,3);                   //“2 秒漂洗工作循环”进行 3 次
    while(1);
}
```

```
void multy_rinse(uchar n2,uchar circle2)          //自动漂洗函数
{
    uchar t2;
    for(t2=0;t2<circle2;t2++)          //循环 circle2 次
    {
        rinse(n2);                     //单次漂洗
    }
}
```

4.2.3 step3：脱水控制模块设计

本步骤主要完成的具体任务是采用“脱水工作”方式，即电机全速正转 5 秒后关闭，保持惯性直至停止，程序设计流程图如图 1.6 所示。

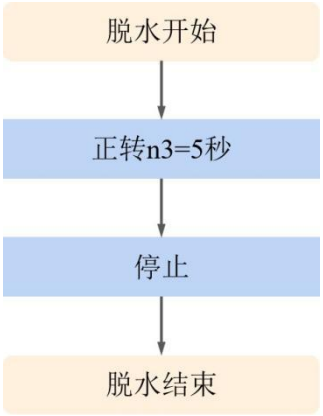


图 1.6 脱水模块程序设计流程图

针对本步骤的设计任务，调用电机正转函数即可实现 5 秒脱水控制。将脱水工作时间设为变量 n3，通过 if 语句循环延时 1000ms 函数调整。同时，在主函数中调用脱水函数，根据功能要求设定脱水工作时间 n3 为 5 秒。相应程序如下：

```
...
void spin_dry(uchar n3);               //脱水函数声明

void main()                            //主函数
{
    multy_wash(3,3);                   //“3 秒洗涤工作循环”进行 3 次
    multy_rinse(2,3);                  //“2 秒漂洗工作循环”进行 3 次
    spin_dry(5);                       //“脱水工作”进行 5 秒
    while(1);
}

void spin_dry(uchar n3)                //脱水函数
{
```

```
    uchar k3;
    fwd();                          //电机正转
    for(k3=0;k3<n3;k3++)            //延时 n3 秒
    {
        Delay1000ms();
    }
    stop();                         //电机停止
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#define uchar unsigned char
sbit motor_a=P2^6;                  //电机端口 P2.6 定义
sbit motor_b=P2^7;                  //电机端口 P2.7 定义
void Delay10ms();                   //延时 10ms 函数声明
void Delay500ms();                  //延时 500ms 函数声明
void Delay1000ms();                 //延时 1000ms 函数声明
void fwd();                         //电机正转函数声明
void back();                        //电机反转函数声明
void stop();                        //电机停止函数声明
void wash(uchar n1);                //洗涤函数声明
void multy_wash(uchar n1,uchar circle1);        //自动洗涤函数声明
void rinse(uchar n2);               //漂洗函数声明
void multy_rinse(uchar n2,uchar circle2);       //自动漂洗函数声明
void spin_dry(uchar n3);            //脱水函数声明

void main()                         //主函数
{
    multy_wash(3,3);                //“3 秒洗涤工作循环”进行 3 次
    multy_rinse(2,3);               //“2 秒漂洗工作循环”进行 3 次
    spin_dry(5);                    //“脱水工作”进行 5 秒
    while(1);
}

void wash(uchar n1)                 //洗涤函数
{
    uchar k1;
    fwd();                          //电机正转
    for(k1=0;k1<n1;k1++)            //延时 n1*1000ms
    {
```

```
        Delay1000ms();
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
    back();                         //电机反转
    for(k1=0;k1<n1;k1++)            //延时 n1*1000ms
    {
        Delay1000ms();
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
}

void multy_wash(uchar n1,uchar circle1)          //自动洗涤函数
{
    uchar t1;
    for(t1=0;t1<circle1;t1++)       //循环 circle1 次
    {
        wash(n1);                   //单次洗涤
    }
}

void rinse(uchar n2)                //漂洗函数
{
    uchar k2;
    n2=50*n2;                       //将工作时间转化为循环次数
    for(k2=0;k2<n2;k2++)
    {
        fwd();                      //电机正转
        Delay10ms();                //延时 10ms
        stop();                     //电机停止
        Delay10ms();                //延时 10ms
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
    for(k2=0;k2<n2;k2++)
    {
        back();                     //电机反转
        Delay10ms();                //延时 10ms
        stop();                     //电机停止
        Delay10ms();                //延时 10ms
    }
    stop();                         //电机停止
    Delay500ms();                   //延时 500ms
```

```
}

void multy_rinse(uchar n2,uchar circle2)              //自动漂洗函数
{
    uchar t2;
    for(t2=0;t2<circle2;t2++)             //循环 circle2 次
    {
        rinse(n2);                        //单次漂洗
    }
}

void spin_dry(uchar n3)                   //脱水函数
{
    uchar k3;
    fwd();                                //电机正转
    for(k3=0;k3<n3;k3++)                  //延时 n3 秒
    {
        Delay1000ms();
    }
    stop();                               //电机停止
}

void fwd()                                //电机正转函数
{
    motor_a=1;
    motor_b=0;
}
void back()                               //电机反转函数
{
    motor_a=0;
    motor_b=1;
}
void stop()                               //电机停止函数
{
    motor_a=1;
    motor_b=1;
}

void Delay10ms()                          //延时 10ms 函数
{
    unsigned char i, j;
    i = 18;
    j = 235;
    do
```

```
    {
        while (--j);
    } while (--i);
}

void Delay500ms()                    //延时 500ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 4;
    j = 129;
    k = 119;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}

void Delay1000ms()                   //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

5.本章小结

本章设计开发了一种全自动洗衣机控制系统，主要涉及电机模块的综合应用，包括直流电机的正反停转控制和 PWM 调速控制，通过对这两者的具体参数设置和交叉关联运用，能够实现洗衣机控制系统洗涤、漂洗、脱水模块不同的功能要

求。同时，本章从具体的任务要求出发，展示了控制系统设计的一系列流程，尤其是控制系统的软件设计部分，如何确定系统软件架构，并以此为依据规划 step by step 程序设计步骤，如何通过流程图厘清程序设计思路，并根据设计思路开展程序设计实践。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用电机模块的控制功能，并能够独立完成顺序结构控制系统的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，增加全自动洗衣机的人机交互功能，要求能够通过相应按键切换洗衣模式，并在洗衣过程任意时刻打开盖子即停止工作，具体功能自拟。

一种国旗升降控制系统开发

1.学习目标

1. 能根据控制系统任务要求完成独立按键的输入判断和直流电机的输出控制；
2. 能使用标志位确定系统软件架构，制定分步实施方案；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

按键可以向单片机输入信号，发送命令，是单片机应用中人机交互的主要手段。其中 AT89C52 单片机又可以分为独立按键和矩阵按键两种结构，独立按键的特点是每个按键各连接一个 I/O 接口，识别容易，判断简单，矩阵按键的特点是由行列结构交叉构成，能够使用较少的 I/O 接口控制更多的按键。在所需按键数目较少的情况下，使用独立按键输入控制直流电机输出，以实现单片机完整的输入输出控制。本章提出一例“设计一种国旗升降控制系统”的课题需求，学习独立按键的检测原理和输入判断，并掌握单片机使用标志位简化软件架构、实现控制功能的程序设计方法。

任务题目：设计一种国旗升降控制系统

功能要求：该系统能够通过按下不同按键，控制电机旋转，带动国旗匀速上升、到顶停止、匀速下降、到底停止，实现国旗升降装置的远距按键控制。同时，在系统运行过程中如果遇到突发情况，能够按下暂停按键停止运行。分别设定升旗按键、降旗按键、暂停按键、上限位传感器、下限位传感器，系统独立按键功能配置如表 2.1 所示。

表 2.1 系统独立按键功能配置表

上限位传感器 K5			下限位传感器 K8
升旗按键 K1	暂停按键 K2		降旗按键 K4

具体控制要求如下：

1.升旗功能。按下升旗按键，升降电机以 50%占空比的速度正转，带动国旗匀速上升，到达旗杆顶部触碰上限位传感器，升降电机停止，国旗停止上升。国旗处于底部状态时只响应升旗按键，国旗处于上升状态时只响应上限位传感器。

2.降旗功能。按下降旗按键，升降电机以 50%占空比的速度反转，带动国旗匀速下降，到达旗杆底部触碰下限位传感器，升降电机停止，国旗停止下降。国旗处于顶部状态时只响应降旗按键，国旗处于下降状态时只响应下限位传感器。

3.暂停功能。系统在任意工作状态，如果遇到突发情况，可以按下暂停按键，升降电机停止，国旗停止升降，此时升/降按键、上/下限位传感器均不响应。当排除突发情况，可以再次按下暂停按键，系统仍然按照暂停前的升降状态工作。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输入模块

单片机输入模块包括独立按键控制模块，主要用于系统运行过程中升旗、降旗、暂停、限位各控制功能的人机操作。已知独立按键每个按键各连接一个 I/O 接口，当按键未按下时，对应的 I/O 接口表现为高电平，当按键按下时，对应的 I/O 接口则变更为低电平。因此，单片机通过判断 I/O 接口的电平状态，能够识别对应按键是否按下或松开。独立按键接口电路如图 2.1 所示，在开发板上分别占用 P1.4、P1.5、P1.6、P1.7、P3.2、P3.3、P3.4、P3.5 引脚。以按键 K1 为例，当 K1 按下时 P3.2=0，当 K1 松开时 P3.2=1。

此外，单片机使用的按键为机械触点按键。由于机械触点的弹性振动，这种按键在闭合和断开的瞬间会产生抖动，抖动时间取决于按键的机械特性，一般为 5~10ms。按键抖动会造成单片机对操作次数的判断错误，需要进行消抖处理。常用的按键消抖方法是使用软件延时消抖，这里介绍另一种按键消抖方法，通过判断按键的上一状态和当前状态来确认按键确实按下后松开，保证单片机对一次

按键动作只确认一次按键有效。以按键 K1 为例，当上一状态 K1_last=0 且当前状态 K1_now=1 时，表明 K1 按下后松开为一次按键有效。

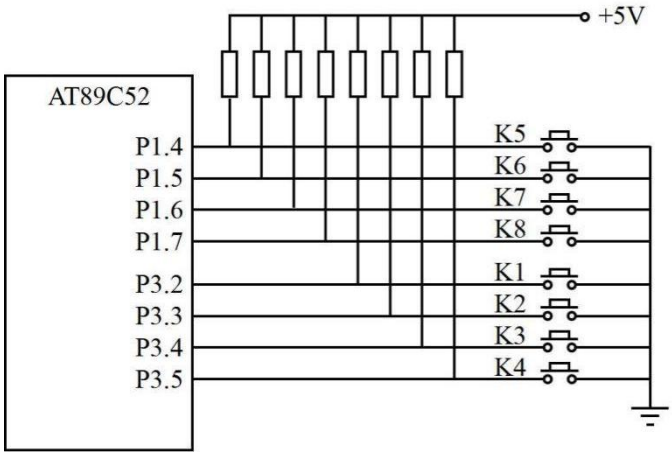


图 2.1 独立按键接口电路图

3.2 单片机输出模块

单片机输出模块包括直流电机控制模块，主要用于国旗上升和下降的动力驱动。该功能由单片机输出信号控制电机驱动，对旋转角度的控制要求不高，可以选择直流电机及其驱动模块的控制方案。同时该模块的选型应根据实际控制装置的要求进行，本例不作展开。直流电机的控制原理已在第一章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P2.6/P2.7 引脚。

3.3 单片机内部配置

系统端口及硬件配置如表 2.2 所示。

表 2.2 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P3.2	升旗按键
2	P3.5	降旗按键
3	P3.3	暂停按键
4	P1.4	上限位传感器
5	P1.7	下限位传感器
6	P2.6/P2.7	直流电机

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“升降控制”和“暂停控制”两个模块。

“升降控制”模块是该系统的核心模块，用户按照功能顺序依次按下不同的按键驱动电机以相应的状态工作：按下升旗按键，电机以 50%占空比的速度正转；触碰上限位传感器，电机停止；按下降旗按键，电机以 50%占空比的速度反转；触碰下限位传感器，电机停止。为了明确系统输入输出控制之间的对应关系，引入标志位的概念，使用标志位界定系统工作的当前状态。由上可知，可以将系统工作的电机状态分为四种，因此设定 unsigned char 类型的电机标志位 motor_flag。其初始状态为 motor_flag=0，此时输出使电机停止，表现为国旗停于底部；按下升旗按键，输入置 motor_flag=1，此时输出使电机以 50%占空比的速度正转，表现为国旗匀速上升；按下上限位键，输入置 motor_flag=2，此时输出使电机停止，表现为国旗停于顶部；按下降旗按键，输入置 motor_flag=3，此时输出使电机以 50%占空比的速度反转，表现为国旗匀速下降；按下下限位键，回到初始状态 motor_flag=0，如此循环进行。

“暂停控制”模块不再涉及顺序控制，用户按照功能要求按下暂停按键切换系统的启停状态：首次按下暂停按键，系统停止工作；再次按下暂停按键，系统按照暂停前的状态工作。同样使用标志位界定其当前状态，设定 bit 类型的暂停标志位 pause_flag。首次按下暂停按键，输入置 pause_flag=1，此时输出为系统停止；再次按下暂停按键，输入置 pause_flag=0，此时输出为系统工作。

系统标志位对应关系如表 2.3 所示。

表 2.3 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
motor_flag	0	电机停止
	1	电机 50%速度正转
	2	电机停止
	3	电机 50%速度反转
pause_flag	0	系统工作
	1	系统停止

根据标志位确定软件架构，将“升降控制”模块置于“暂停控制”模块的工作状态下。系统软件架构如图 2.2 所示。

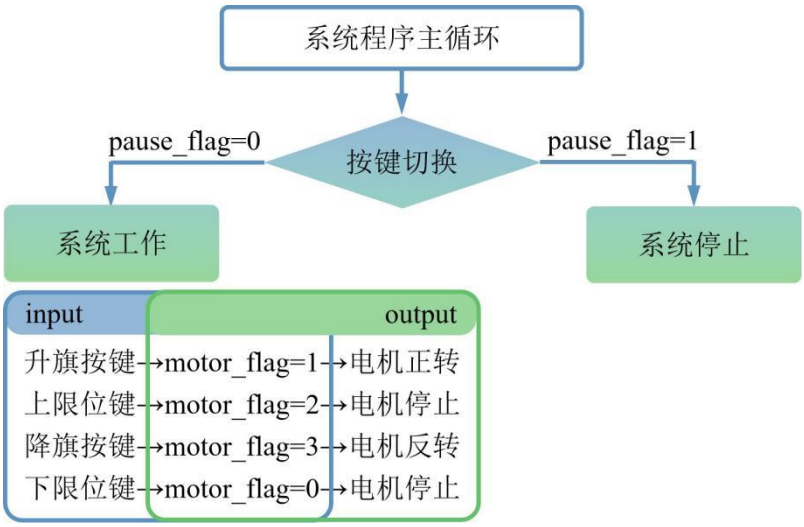


图 2.2 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由两个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 2.4 所示。

表 2.4 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	升降控制模块设计	独立按键和直流电机的联动控制
Step 2	暂停控制模块设计	标志位划分系统状态的应用

4.2.1 step1：升降控制模块设计

本步骤主要完成的具体任务是通过按下不同按键，控制电机旋转，带动国旗匀速上升、到顶停止、匀速下降、到底停止。根据系统软件架构分析，将该模块分为按键输入部分（input）和电机输出部分（output），使用电机标志位 motor_flag 界定系统工作的四个状态，程序设计流程图如图 2.3 所示。需要注意的是，系统上电后需要实时检测各个按键并控制直流电机，因此将按键输入和电机输出放置在主函数的 while(1)循环中执行。另外，直流电机的正反停转控制和 PWM 调速控制的程序实现已在第一章案例中详细讲解，此处不作赘述。

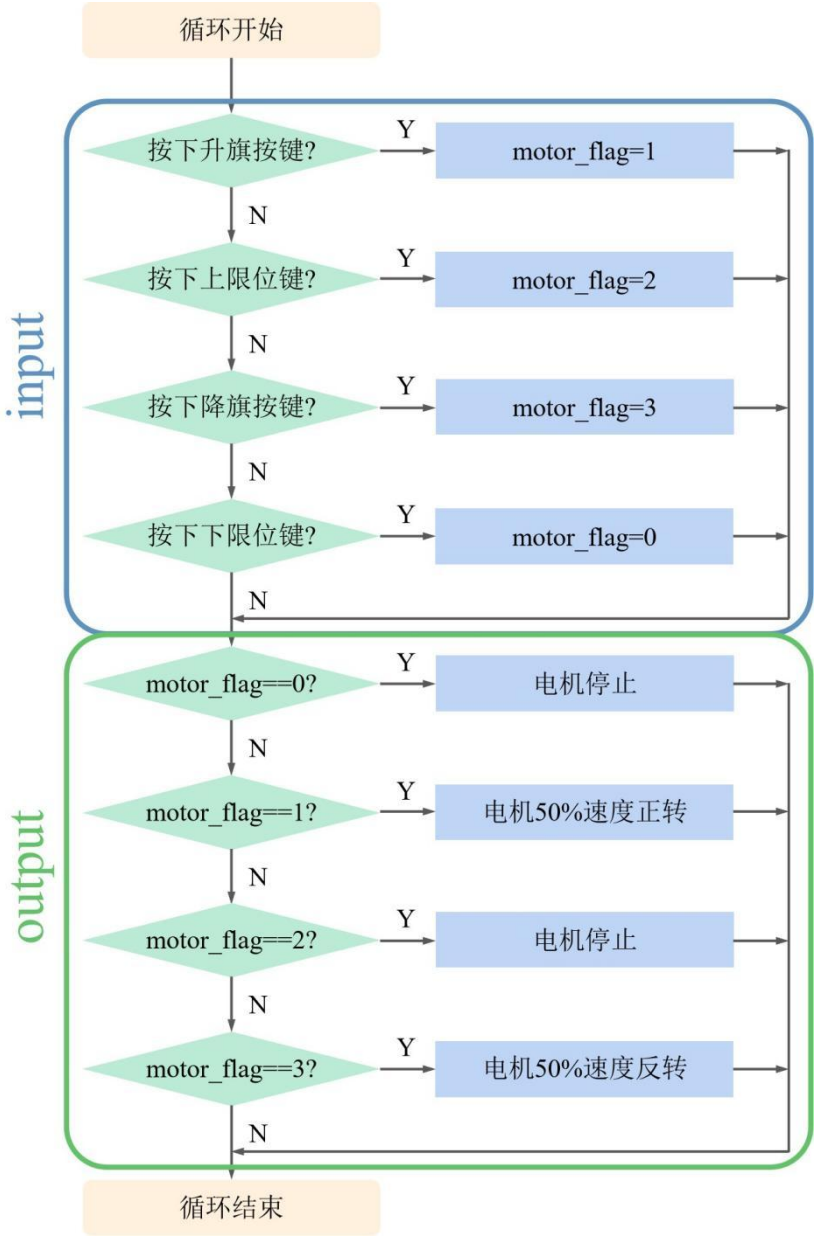


图 2.3 升降模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现按键输入部分，检测四个按键是否按下并置对应的电机标志位，然后在此基础上实现电机输出部分，根据当前的电机标志位控制电机转停。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，定义 up_key、dn_key、tp_sensor、bm_sensor 分别为升旗按键端口 P3.2、降旗按键端口 P3.5、上限位传感器端口 P1.4、下限位传感器端口 P1.7，定义 motor_flag 为电机标志位，根据升降模块程序设计流程图建立按键输入函数，并声明程序中使用的所有函数。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#define uchar unsigned char
sbit up_key=P3^2;                    //升旗按键端口 P3.2 定义
sbit dn_key=P3^5;                    //降旗按键端口 P3.5 定义
sbit tp_sensor=P1^4;                 //上限位传感器端口 P1.4 定义
sbit bm_sensor=P1^7;                 //下限位传感器端口 P1.7 定义
uchar motor_flag;                    //电机标志位定义
void input();                        //按键输入函数声明

void input()                         //按键输入函数
{
    if(up_key==0&&motor_flag==0)              //底部状态下按下升旗按键
    {
        motor_flag=1;                //置电机标志位为 1
    }
    else if(tp_sensor==0&&motor_flag==1)      //上升状态下按下上限位键
    {
        motor_flag=2;                //置电机标志位为 2
    }
    else if(dn_key==0&&motor_flag==2)         //顶部状态下按下降旗按键
    {
        motor_flag=3;                //置电机标志位为 3
    }
    else if(bm_sensor==0&&motor_flag==3)      //下降状态下按下下限位键
    {
        motor_flag=0;                //置电机标志位为 0
    }
}
```

（2）在此基础上继续完成电机输出部分，定义 motor_a/motor_b 为电机端口 P2.6/P2.7，根据升降模块程序设计流程图建立电机输出函数，其中电机转停控制

通过建立并调用电机正转、反转、停止函数来实现，所涉及的延时函数可以在STC 软件的软件延时计算器中得到。同时，在主函数的 while(1)循环中依次调用按键输入函数和电机输出函数。相应程序如下：

```
...
sbit motor_a=P2^6;                  //电机端口 P2.6 定义
sbit motor_b=P2^7;                  //电机端口 P2.7 定义
void Delay10ms();                   //延时 10ms 函数声明
void fwd();                         //电机正转函数声明
void back();                        //电机反转函数声明
void stop();                        //电机停止函数声明
void output();                      //电机输出函数声明

void main()                         //主函数
{
    while(1)
    {
        input();                    //按键输入
        output();                   //电机输出
    }
}

void output()                       //电机输出函数
{
    if(motor_flag==0)               //电机标志位为 0
    {
        stop();                     //电机停止
    }
    else if(motor_flag==1)          //电机标志位为 1
    {
        fwd();                      //电机正转
        Delay10ms();                //延时 10ms
        stop();                     //电机停止
        Delay10ms();                //延时 10ms
    }
    else if(motor_flag==2)          //电机标志位为 2
    {
        stop();                     //电机停止
    }
    else if(motor_flag==3)          //电机标志位为 3
    {
        back();                     //电机反转
        Delay10ms();                //延时 10ms
        stop();                     //电机停止
```

```
            Delay10ms();                    //延时 10ms
        }
    }

    void fwd()                              //电机正转函数
    {
        motor_a=1;
        motor_b=0;
    }
    void back()                             //电机反转函数
    {
        motor_a=0;
        motor_b=1;
    }
    void stop()                             //电机停止函数
    {
        motor_a=1;
        motor_b=1;
    }

    void Delay10ms()                        //延时 10ms 函数
    {
        unsigned char i, j;
        i = 18;
        j = 235;
        do
        {
            while (--j);
        } while (--i);
    }
```

4.2.2 step2：暂停控制模块设计

本步骤主要完成的具体任务是在系统运行过程中能够按下暂停按键停止运行。根据系统软件架构分析，使用暂停标志位 pause_flag 界定系统启停的两个状态，程序设计流程图如图 2.4 所示。可以发现暂停控制模块嵌套于升降控制模块之外，系统工作状态下执行升降控制模块程序，按照升降功能要求正常运行，系统停止状态下直接停止直流电机，此时升/降按键、上/下限位传感器均不响应。

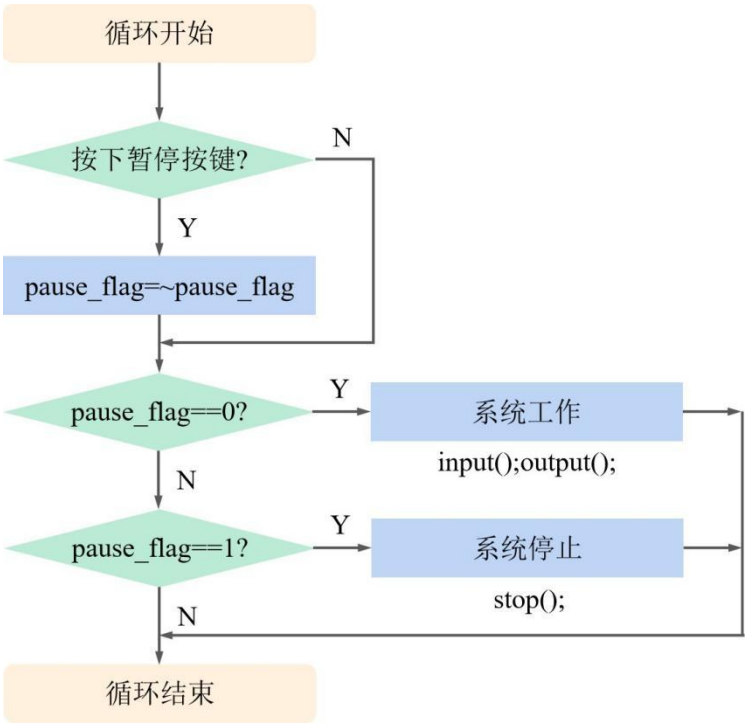


图 2.4 暂停模块程序设计流程图

针对本步骤的设计任务，设计思路同样应考虑检测暂停按键是否按下并置对应的暂停标志位，然后根据当前的暂停标志位控制系统启停。

定义 pause 为暂停按键端口 P3.3，并初始化暂停按键键值，定义 pause_flag 为暂停标志位。在主函数的 while(1)循环中使用按键消抖方法刷新暂停按键键值，检测按键按下后松开的单次按键有效动作，当暂停按键确实按下后松开，取反暂停标志位以切换系统状态：当暂停标志位为 0 时，此时系统工作，依次调用按键输入函数和电机输出函数；当暂停标志位为 1 时，此时系统停止，调用电机停止函数。相应程序如下：

```
...
sbit pause=P3^3;                    //暂停按键端口 P3.3 定义
bit pause_last=1,pause_now=1;       //暂停按键键值初始化
bit pause_flag;                     //暂停标志位定义

void main()                         //主函数
{
    while(1)
    {
        pause_last=pause_now;
        pause_now=pause;            //暂停按键键值刷新
        if(pause_last==0&&pause_now==1)        //暂停按键按下后松开
        {
```

```
            pause_flag=~pause_flag;   //暂停标志位取反
        }
        if(pause_flag==0)             //暂停标志位为 0
        {
            input();                  //按键输入
            output();                 //电机输出
        }
        else if(pause_flag==1)        //暂停标志位为 1
        {
            stop();                   //电机停止
        }
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#define uchar unsigned char
sbit motor_a=P2^6;                    //电机端口 P2.6 定义
sbit motor_b=P2^7;                    //电机端口 P2.7 定义
sbit up_key=P3^2;                     //升旗按键端口 P3.2 定义
sbit dn_key=P3^5;                     //降旗按键端口 P3.5 定义
sbit tp_sensor=P1^4;                  //上限位传感器端口 P1.4 定义
sbit bm_sensor=P1^7;                  //下限位传感器端口 P1.7 定义
sbit pause=P3^3;                      //暂停按键端口 P3.3 定义
bit pause_last=1,pause_now=1;         //暂停按键键值初始化
bit pause_flag;                       //暂停标志位定义
uchar motor_flag;                     //电机标志位定义
void Delay10ms();                     //延时 10ms 函数声明
void fwd();                           //电机正转函数声明
void back();                          //电机反转函数声明
void stop();                          //电机停止函数声明
void input();                         //按键输入函数声明
void output();                        //电机输出函数声明

void main()                           //主函数
{
    while(1)
    {
        pause_last=pause_now;
        pause_now=pause;              //暂停按键键值刷新
```

```
        if(pause_last==0&&pause_now==1)         //暂停按键按下后松开
        {
            pause_flag=~pause_flag;   //暂停标志位取反
        }
        if(pause_flag==0)               //暂停标志位为 0
        {
            input();                    //按键输入
            output();                   //电机输出
        }
        else if(pause_flag==1)          //暂停标志位为 1
        {
            stop();                     //电机停止
        }
    }
}

void input()                            //按键输入函数
{
    if(up_key==0&&motor_flag==0)                //底部状态下按下升旗按键
    {
        motor_flag=1;               //置电机标志位为 1
    }
    else if(tp_sensor==0&&motor_flag==1)        //上升状态下按下上限位键
    {
        motor_flag=2;               //置电机标志位为 2
    }
    else if(dn_key==0&&motor_flag==2)           //顶部状态下按下降旗按键
    {
        motor_flag=3;               //置电机标志位为 3
    }
    else if(bm_sensor==0&&motor_flag==3)        //下降状态下按下下限位键
    {
        motor_flag=0;               //置电机标志位为 0
    }
}

void output()                           //电机输出函数
{
    if(motor_flag==0)               //电机标志位为 0
    {
        stop();                     //电机停止
    }
    else if(motor_flag==1)          //电机标志位为 1
    {
```

```
            fwd();                          //电机正转
            Delay10ms();                    //延时 10ms
            stop();                         //电机停止
            Delay10ms();                    //延时 10ms
        }
        else if(motor_flag==2)              //电机标志位为 2
        {
            stop();                         //电机停止
        }
        else if(motor_flag==3)              //电机标志位为 3
        {
            back();                         //电机反转
            Delay10ms();                    //延时 10ms
            stop();                         //电机停止
            Delay10ms();                    //延时 10ms
        }
}

void fwd()                                  //电机正转函数
{
    motor_a=1;
    motor_b=0;
}
void back()                                 //电机反转函数
{
    motor_a=0;
    motor_b=1;
}
void stop()                                 //电机停止函数
{
    motor_a=1;
    motor_b=1;
}

void Delay10ms()                            //延时 10ms 函数
{
    unsigned char i, j;
    i = 18;
    j = 235;
    do
    {
        while (--j);
    } while (--i);
}
```

5.本章小结

本章设计开发了一种国旗升降控制系统，使用电机模块作为系统运动的输出模块，在此基础上增加按键模块作为系统交互的输入模块，实现从用户按下按键到电机带动国旗的完整控制过程。同时，本章首次介绍了标志位的使用，通过标志位界定系统当前状态，确定系统软件架构，并以此为依据规划 step by step 程序设计步骤。本章展示了控制系统设计中的一个重要思路，即系统输入仅改变标志位数值，再根据标志位数值执行相应系统输出，这种思路能够帮助梳理具有复杂功能、多层级多任务的控制系统，这在后面章节的案例中将有所体现。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用按键模块和电机模块，并能够独立完成使用标志位实现控制功能的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，使用单键复用整合升旗、降旗、上限位、下限位四个按键的功能，同时使用液晶显示屏实时刷新显示系统工作的当前状态，具体功能自拟。

一种导弹发射控制系统的密码模块开发

1.学习目标

1. 能根据控制系统任务要求完成独立按键的密码输入和液晶显示的实时刷新；
2. 能使用标志位划分层级系统状态，制定分步实施方案；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

为了更加直观地展示控制系统运行的各项信息数据，可以使用液晶显示屏实时刷新显示控制系统界面。至此，直流电机、独立按键、液晶显示这三个模块的联动控制基本能够涵盖综合控制系统的功能要求。本书以单片机系统最常见的字符型液晶显示模块 1602LCD 为例介绍液晶显示相关控制功能，其头文件见附录“1602LCD.h”。

本章提出一例“设计一种导弹发射控制系统的密码模块”的课题需求，学习液晶显示 1602LCD 的调用函数和使用技巧，并掌握单片机层级结构控制系统的程序设计方法。

任务题目：设计一种导弹发射控制系统的密码模块

功能要求：该系统在初始界面下能够通过按下数字键输入密码并判断密码，密码正确则执行导弹发射或密码重置，密码错误则刷新显示界面甚至锁定系统，其中液晶显示 1602LCD 一直跟随系统运行实时刷新显示。分别设定 1~6 数字键、重置按键、发射按键，系统独立按键功能配置如表 3.1 所示。

表 3.1 系统独立按键功能配置表

1 数字键 K5	2 数字键 K6	3 数字键 K7	重置按键 K8
4 数字键 K1	5 数字键 K2	6 数字键 K3	发射按键 K4

具体控制要求如下：

1.初始界面显示。系统上电后在 1602LCD 上显示宣传标语“Ke Ji Qiang Guo”、

“Shi Gan Bao Guo”。1 秒后清屏，进入主控初始界面：第 1 行显示系统名称“DONGFENG”，第 2 行显示密码输入界面“input code: ___”，提示用户输入 3 位密码。

2.密码输入功能。按下 1~6 数字键输入 3 位密码，如果密码正确，显示“READY!”信息，如果密码错误，刷新密码输入界面。如果密码输入 3 次仍不正确，则系统锁定，显示“LOCK!!”信息。同时按下重置按键和 1 数字键解锁，回到密码输入界面。

3.导弹发射功能。当密码输入正确，按下发射按键，继电器导通（控制点火装置发射导弹），1 秒后自动断开，回到密码输入界面。

4.密码重置功能。当密码输入正确，按下重置按键，第 2 行显示密码重置界面“set code: ___”，系统显示并保存随后按下的 3 位数字键作为新密码，显示“SAVE OK!”信息。再次按下重置按键，回到密码输入界面，此时输入新密码进入系统。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输入模块

单片机输入模块包括独立按键控制模块，主要用于系统运行过程中密码输入、密码重置、导弹发射、系统解锁各控制功能的人机操作。单片机通过判断 I/O 接口的电平状态，能够识别对应按键是否按下或松开，其控制原理已在第二章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P1.4、P1.5、P1.6、P1.7、P3.2、P3.3、P3.4、P3.5 引脚。

3.2 单片机输出模块

单片机输出模块包括继电器控制模块和液晶显示控制模块。

（1）继电器控制模块主要用于导弹发射功能的点火装置控制。该模块由PNP 型三极管和继电器线圈组成，单片机输出一个信号使得三极管处于导通或断开的状态，继而控制继电器线圈得电吸合或断电释放。继电器接口电路如图 3.1 所示，在开发板上占用 P3.7 引脚。当 P3.7=0 时继电器导通，当 P3.7=1 时继电器断开。

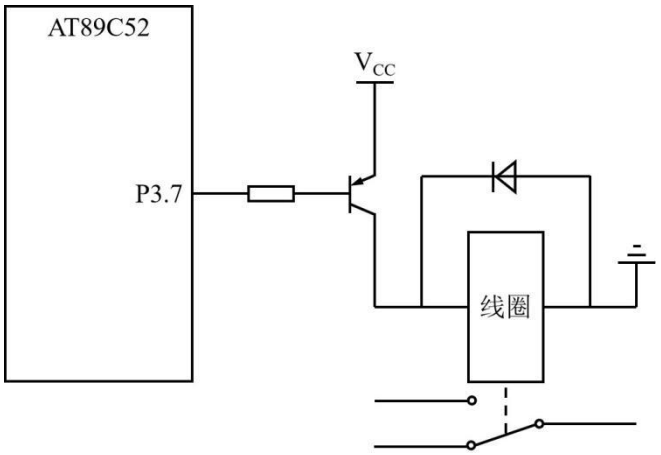


图 3.1 继电器接口电路图

（2）液晶显示控制模块主要用于系统运行过程中各项信息数据的实时刷新显示。该模块包括 8 个数据引脚、3 个控制引脚和 3 个电源引脚，单片机通过数据引脚和控制引脚输出数据和命令至 1602LCD，从而实现相应的数字、字母、符号显示。液晶显示接口电路如图 3.2 所示，在开发板上占用 P0+P2.0/P2.1/P2.2 引脚，其中 P0.0~P0.7 连接数据引脚，P2.0/P2.1/P2.2 连接控制引脚。

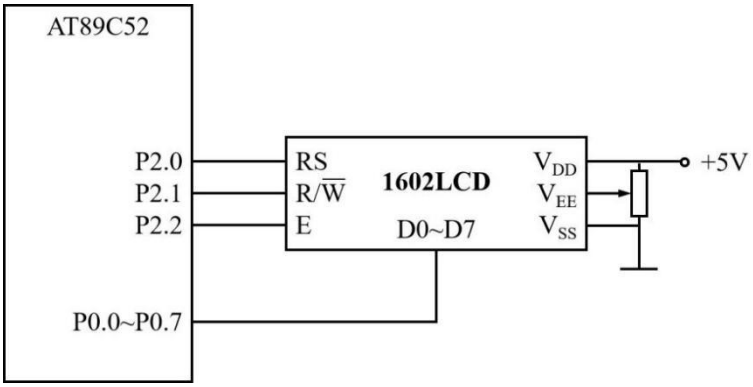


图 3.2 液晶显示接口电路图

3.3 单片机内部配置

系统端口及硬件配置如表 3.2 所示。

表 3.2 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P1.4	1 数字键
2	P1.5	2 数字键
3	P1.6	3 数字键
4	P3.2	4 数字键
5	P3.3	5 数字键
6	P3.4	6 数字键
7	P1.7	重置按键
8	P3.5	发射按键
9	P3.7	继电器
10	P0+P2.0/P2.1/P2.2	液晶显示

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

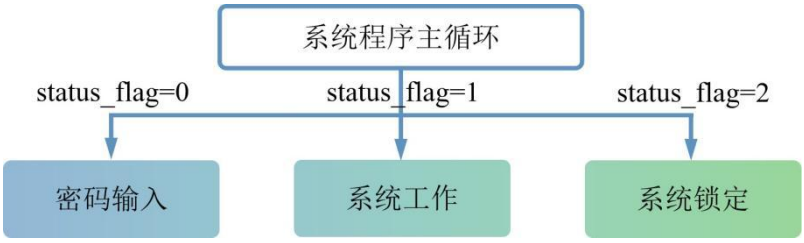
分析控制系统的功能要求，可以发现系统在各个状态下的按键输入和液晶显示均不相同，如果在切换状态的同时直接作相应变化，软件程序会过于繁琐复杂。这里延续第二章的程序设计思路，控制系统切换状态仅改变标志位数值，再根据标志位数值执行不同程序片段。

使用标志位梳理控制功能，确定软件架构，其关键在于界定控制系统的当前状态。需要注意的是，一个控制系统的状态划分会有多种方案，结合实际情况选择合理方案即可。

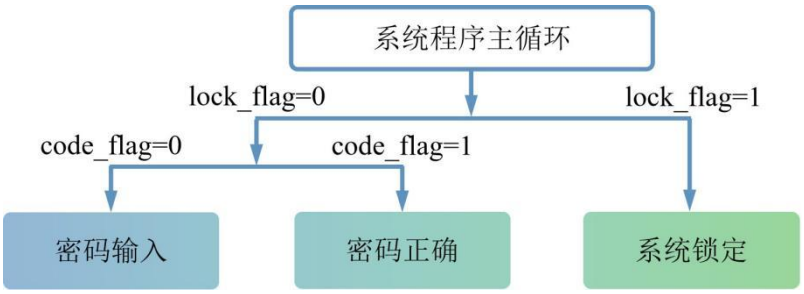
例如，可以将本例控制功能罗列为“密码输入”、“系统工作”、“系统锁定”三个同一级别的模块，设定 unsigned char 类型的状态标志位 status_flag，如图 3.3(a)

所示：系统上电后默认 status_flag=0 进入“密码输入”模块，此时可执行密码输入操作，如果密码正确，则置 status_flag=1 进入“系统工作”模块，此时可执行导弹发射或密码重置操作，如果密码错误 3 次，则置 status_flag=2 进入“系统锁定”模块，此时可执行系统解锁操作。

也可以将本例控制功能罗列为“系统工作”/“系统锁定”和“密码输入”/“密码正确”两组不同级别的模块，分别设定 bit 类型的锁定标志位 lock_flag 和密码标志位 code_flag，并将密码模块置于系统模块的工作状态下，如图 3.3(b)所示：系统上电后默认 lock_flag=0 进入“系统工作”模块，此时系统正常运行。默认 code_flag=0 进入“密码输入”模块，此时可执行密码输入操作，如果密码正确，则置 code_flag=1 进入“密码正确”模块，此时可执行导弹发射或密码重置操作，如果密码错误 3 次，则置 lock_flag=1 进入“系统锁定”模块，此时可执行系统解锁操作。



(a)



(b)

图 3.3 系统状态划分示意图

选择图 3.3(b)所示方案，再次细化各个模块下的具体功能和状态切换。在“密码输入”模块下，涉及的状态切换包括密码正确置 code_flag=1 和密码错误 3 次置 lock_flag=1；在“密码正确”模块下，涉及的状态切换包括导弹发射执行结束置 code_flag=0 和密码重置执行结束置 code_flag=0；在“系统锁定”模块下，涉及的状态切换为系统解锁置 lock_flag=0。

系统标志位对应关系如表 3.3 所示。

表 3.3 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
lock_flag	0	系统工作
	1	系统锁定
code_flag	0	密码输入
	1	密码正确

系统软件架构如图 3.4 所示。

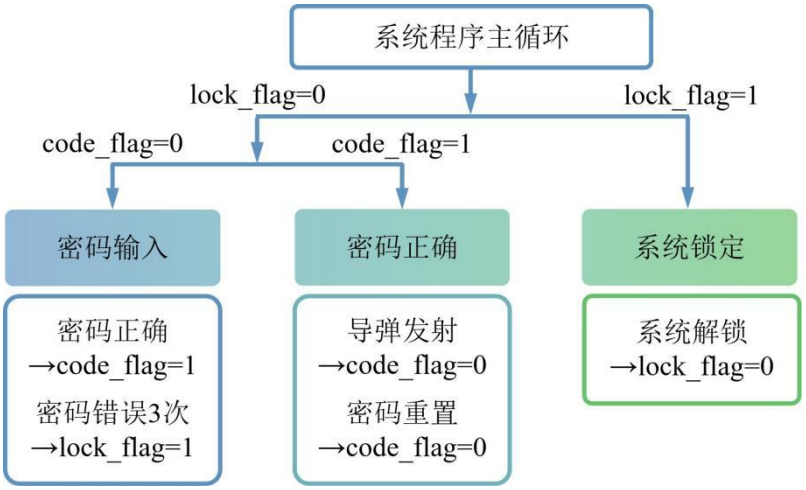


图 3.4 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由四个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 3.4 所示。

表 3.4 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	系统程序框架设计	标志位划分系统状态的应用
Step 2	密码输入模块设计	系统状态切换及界面刷新
Step 3	密码正确模块设计	系统状态切换及界面刷新
Step 4	系统锁定模块设计	系统状态切换及界面刷新

4.2.1 step1：系统程序框架设计

在进行控制系统各个模块的程序设计之前，搭建系统程序框架，即将软件架构以程序语言的形式体现。此外，系统上电后需要进行初始化相关操作，通常包括初始界面显示、中断相关配置、初始参数设置等，放置在主函数的 while(1)循环前执行。

针对本步骤的设计任务，根据软件架构使用标志位划分系统状态以执行不同模块程序即可，同时根据功能要求调用液晶显示函数实现初始界面显示。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，其中“1602LCD.h”为液晶显示头文件。定义 lock_flag 为锁定标志位，code_flag 为密码标志位：当锁定标志位为 0 时，此时系统工作，根据密码标志位为 0 或 1 分别执行“密码输入”或“密码正确”模块程序；当锁定标志位为 1 时，此时系统锁定，执行“系统锁定”模块程序。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"1602LCD.h"
#define uchar unsigned char
bit lock_flag;                          //锁定标志位定义
bit code_flag;                          //密码标志位定义

void main()                             //主函数
{
    while(1)
    {
        if(lock_flag==0)                //锁定标志位为 0
        {
            if(code_flag==0)            //密码标志位为 0
            {
            }
            else                        //密码标志位为 1
            {
            }
        }
        else                            //锁定标志位为 1
        {
        }
    }
}
```

（2）液晶显示调用函数如表 3.5 所示。其中 lcd_init()为液晶显示初始化函数，在使用液晶显示之前需要进行初始化设置；lcd_clear()为液晶显示清屏函数，通常刷新显示直接进行部分覆盖即可，该函数可以清除屏幕上的所有内容；lcd_pos()为液晶显示位置函数，在显示内容之前需要指定显示位置，1602LCD 共有 2 行 16 列；lcdwrite_sz()为液晶显示数字函数，用于显示 1 位数字；lcdwrite_zm()为液晶显示字母函数，用于显示 1 位字母或符号，并用单引号标注显示内容；lcdwrite_string()为液晶显示字符串函数，用于显示 1 组字符串，并用双引号标注显示内容，也可以另外定义字符串调用显示。

表 3.5 液晶显示调用函数

函数名称	函数说明
lcd_init()	液晶显示初始化
lcd_clear()	液晶显示清屏
lcd_pos(x,y)	液晶显示位置（显示位置第 x 行第 y 列）
lcdwrite_sz(1)	液晶显示数字（显示数字 1）
lcdwrite_zm('a')	液晶显示字母（显示字母 a）
lcdwrite_string("1602LCD")	液晶显示字符串（显示字符串 1602LCD）

定义初始界面显示所需的宣传标语、系统名称、输入界面为 code 型字符串数组：首先进行液晶显示初始化，指定第 1 行第 1 列显示宣传标语 disp1，第 2 行第 1 列显示宣传标语 disp2，然后延时 1000ms 进行液晶显示清屏，指定第 1 行第 1 列显示系统名称 name，第 2 行第 1 列显示输入界面 input，所涉及的延时函数可以在 STC 软件的软件延时计算器中得到。相应程序如下：

```
...
uchar code disp1[]={"Ke Ji Qiang Guo!"};          //宣传标语 disp1 定义
uchar code disp2[]={"Shi Gan Bao Guo!"};          //宣传标语 disp2 定义
uchar code name[]={"      DONGFENG      "};   //系统名称 name 定义
uchar code input[]={"input code: ___ "};          //输入界面 input 定义
void Delay1000ms();                    //延时 1000ms 函数声明

void main()                            //主函数
{
    lcd_init();                        //液晶显示初始化
    lcd_pos(1,1);                      //显示位置第 1 行第 1 列
    lcdwrite_string(disp1);            //显示宣传标语 disp1
```

```
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
    lcdwrite_string(disp2);         //显示宣传标语 disp2
    Delay1000ms();                  //延时 1000ms
    lcd_clear();                    //液晶显示清屏
    lcd_pos(1,1);                   //显示位置第 1 行第 1 列
    lcdwrite_string(name);          //显示系统名称 name
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
    lcdwrite_string(input);         //显示输入界面 input
    while(1)
    {
    }
}

void Delay1000ms()                  //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

4.2.2 step2：密码输入模块设计

本步骤主要完成的具体任务是通过按下数字键输入密码并判断密码，如果密码正确，显示“READY!”信息，如果密码错误，刷新密码输入界面，错误 3 次则系统锁定，显示“LOCK!!”信息。考虑到密码输入功能和密码重置功能都需要实时检测数字键是否按下，将该模块分为按键检测部分（scan_key）和密码输入部分（input_code），以调用 void 型函数的形式进行。程序设计流程图如图 3.5 所示，其中 count 为密码输入次数，error_count 为密码错误次数。

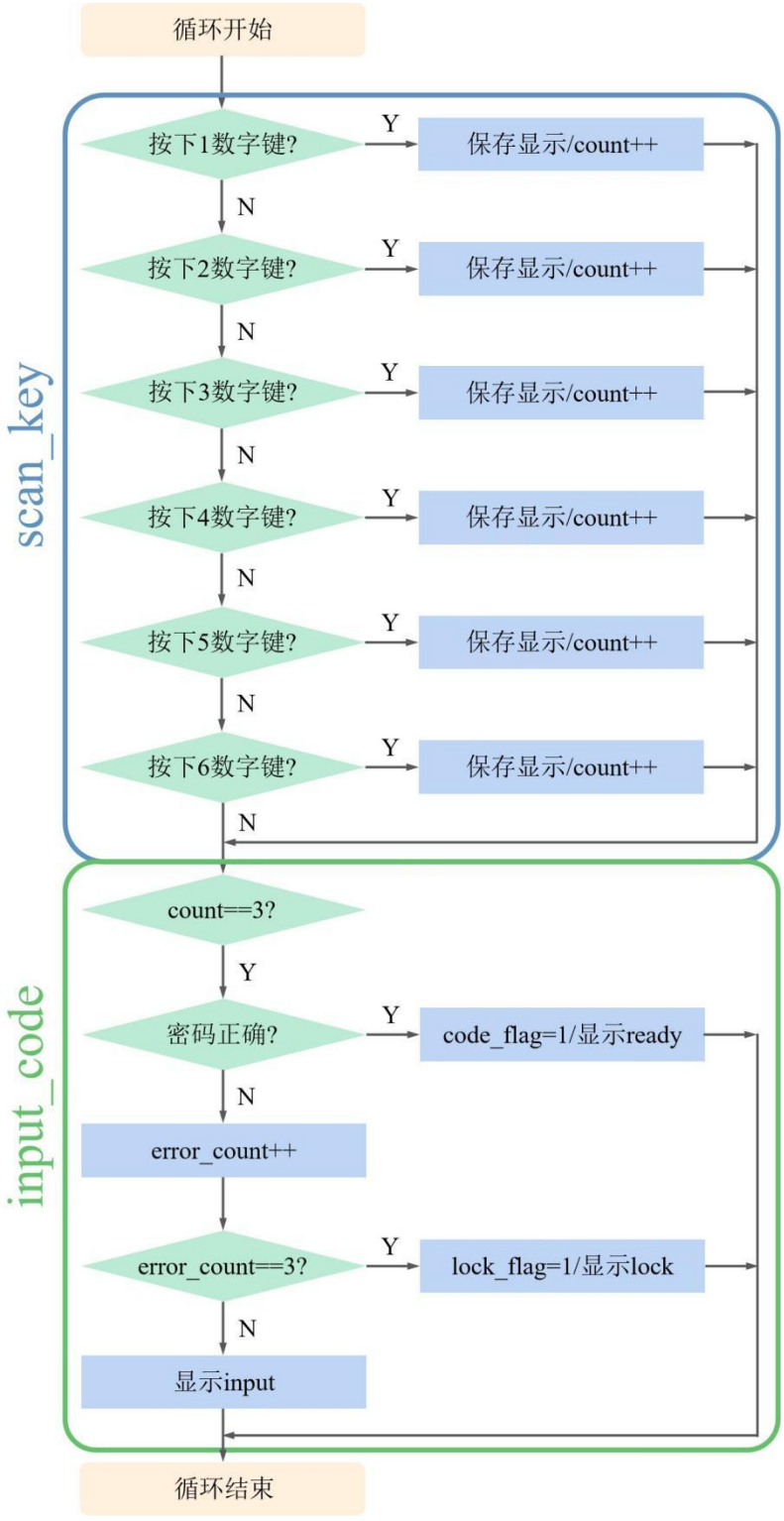


图 3.5 密码输入模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现按键检测部分，检测 1~6 数字键是否按下，保存显示相应数字并对密码输入次数计数，然后在此基础上实现密码输入部分，输入 3 次则比对密码，根据密码是否正确置对应标志位切换系统状态。

（1）按键消抖方法的程序实现已在第二章案例中详细讲解，此处不作赘述。定义 n1、n2、n3、n4、n5、n6 分别为 1~6 数字键端口 P1.4、P1.5、P1.6、P3.2、P3.3、P3.4，并初始化 1~6 数字键键值。定义 code1[]为密码输入数组，定义 count 为密码输入次数，根据密码输入模块程序设计流程图建立按键检测函数，并声明程序中使用的所有函数。相应程序如下：

```
...
sbit n1=P1^4;                          //1 数字键端口 P1.4 定义
bit n1_last=1,n1_now=1;                //1 数字键键值初始化
sbit n2=P1^5;                          //2 数字键端口 P1.5 定义
bit n2_last=1,n2_now=1;                //2 数字键键值初始化
sbit n3=P1^6;                          //3 数字键端口 P1.6 定义
bit n3_last=1,n3_now=1;                //3 数字键键值初始化
sbit n4=P3^2;                          //4 数字键端口 P3.2 定义
bit n4_last=1,n4_now=1;                //4 数字键键值初始化
sbit n5=P3^3;                          //5 数字键端口 P3.3 定义
bit n5_last=1,n5_now=1;                //5 数字键键值初始化
sbit n6=P3^4;                          //6 数字键端口 P3.4 定义
bit n6_last=1,n6_now=1;                //6 数字键键值初始化
uchar code1[]={0,0,0};                 //密码输入数组定义
uchar count;                           //密码输入次数定义
void scan_key();                       //按键检测函数声明

void scan_key()                        //按键检测函数
{
    n1_last=n1_now;
    n1_now=n1;                         //1 数字键键值刷新
    n2_last=n2_now;
    n2_now=n2;                         //2 数字键键值刷新
    n3_last=n3_now;
    n3_now=n3;                         //3 数字键键值刷新
    n4_last=n4_now;
    n4_now=n4;                         //4 数字键键值刷新
    n5_last=n5_now;
    n5_now=n5;                         //5 数字键键值刷新
    n6_last=n6_now;
    n6_now=n6;                         //6 数字键键值刷新
    if(n1_last==0&&n1_now==1)                    //1 数字键按下后松开
    {
        code1[count]=1;                //保存数字 1
        lcd_pos(2,13+count);
        lcdwrite_sz(1);                //显示数字 1
        count++;                       //输入次数计数
```

```
    }
    else if(n2_last==0&&n2_now==1)              //2 数字键按下后松开
    {
        code1[count]=2;                 //保存数字 2
        lcd_pos(2,13+count);
        lcdwrite_sz(2);                 //显示数字 2
        count++;                        //输入次数计数
    }
    else if(n3_last==0&&n3_now==1)              //3 数字键按下后松开
    {
        code1[count]=3;                 //保存数字 3
        lcd_pos(2,13+count);
        lcdwrite_sz(3);                 //显示数字 3
        count++;                        //输入次数计数
    }
    else if(n4_last==0&&n4_now==1)              //4 数字键按下后松开
    {
        code1[count]=4;                 //保存数字 4
        lcd_pos(2,13+count);
        lcdwrite_sz(4);                 //显示数字 4
        count++;                        //输入次数计数
    }
    else if(n5_last==0&&n5_now==1)              //5 数字键按下后松开
    {
        code1[count]=5;                 //保存数字 5
        lcd_pos(2,13+count);
        lcdwrite_sz(5);                 //显示数字 5
        count++;                        //输入次数计数
    }
    else if(n6_last==0&&n6_now==1)              //6 数字键按下后松开
    {
        code1[count]=6;                 //保存数字 6
        lcd_pos(2,13+count);
        lcdwrite_sz(6);                 //显示数字 6
        count++;                        //输入次数计数
    }
}
```

（2）在此基础上继续完成密码输入部分，定义所需的显示内容为 code 型字符串数组，定义 code0[]为正确密码数组（自拟），定义 error_count 为密码错误次数，根据密码输入模块程序设计流程图建立密码输入函数。需要注意的是，系统运行过程中的状态切换往往伴随着参数清零和界面刷新，根据功能要求密码正确、密码错误（1/2 次）、密码错误 3 次都需要刷新显示对应系统界面。同时，在主函

数的 while(1)循环中依次调用按键检测函数和密码输入函数。相应程序如下：

```
...
uchar code ready[]={"      READY!      "};    //显示信息 ready 定义
uchar code lock[]  ={"      LOCK!!      "};    //显示信息 lock 定义
uchar code0[]={1,2,3};                  //正确密码数组定义
uchar error_count;                      //密码错误次数定义
void input_code();                      //密码输入函数声明

void main()                             //主函数
{
    ...
    while(1)
    {
        if(lock_flag==0)                //锁定标志位为 0
        {
            if(code_flag==0)            //密码标志位为 0
            {
                scan_key();             //按键检测
                input_code();           //密码输入
            }
            else                        //密码标志位为 1
            {
            }
        }
        else                            //锁定标志位为 1
        {
        }
    }
}

void input_code()                       //密码输入函数
{
    if(count==3)                        //输入次数为 3
    {
        count=0;                        //输入次数清零
        if(code1[0]==code0[0]&&code1[1]==code0[1]&&code1[2]==code0[2])
                                        //密码正确
        {
            error_count=0;              //错误次数清零
            code_flag=1;                //置密码标志位为 1
            lcd_pos(2,1);
            lcdwrite_string(ready);     //显示信息 ready
        }
        else                            //密码错误
```

```
            {
                error_count++;                  //错误次数计数
                if(error_count==3)              //错误次数为 3
                {
                    error_count=0;              //错误次数清零
                    lock_flag=1;                //置锁定标志位为 1
                    lcd_pos(2,1);
                    lcdwrite_string(lock);      //显示信息 lock
                }
                else                            //错误次数为 1/2
                {
                    lcd_pos(2,1);
                    lcdwrite_string(input);     //显示输入界面 input
                }
            }
        }
    }
```

4.2.3 step3：密码正确模块设计

本步骤涉及到同层级多任务控制，包括导弹发射功能和密码重置功能，这种情况下通过按下发射按键或重置按键进行具体任务执行：按下发射按键，继电器导通 1 秒后断开，回到密码输入界面；按下重置按键，输入数字重置密码并显示“SAVE OK!”信息，再次按下重置按键回到密码输入界面。其中导弹发射功能能够跟随发射按键执行，密码重置功能则需要实时检测数字键，放置在 while(1)循环中执行。因此，设定 bit 类型的重置标志位 set_flag，通过按下重置按键取反标志位，置 set_flag=~set_flag，当 set_flag=1 时进入密码重置状态，当 set_flag=0 时退出密码重置状态。程序设计流程图如图 3.6 所示。

针对本步骤的设计任务，设计思路应考虑分任务完成，首先实现导弹发射部分，检测发射按键是否按下并跟随按键完成导通、延时、断开、置位、显示一系列操作，然后实现密码重置部分，检测重置按键是否按下并取反重置标志位，根据当前的重置标志位执行相应操作和界面刷新。

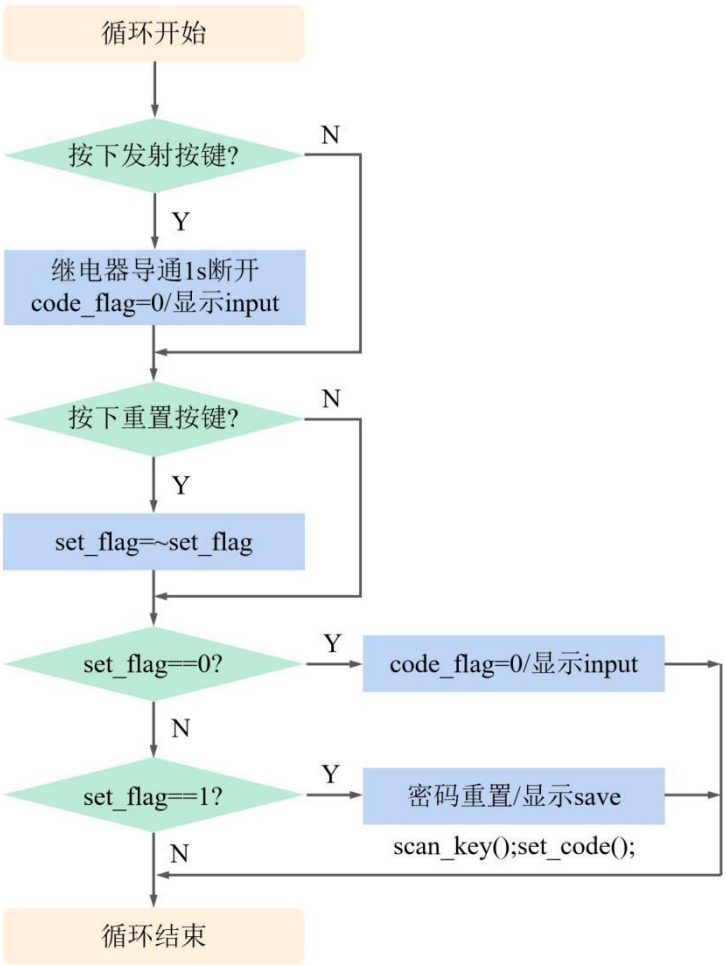


图 3.6 密码正确模块程序设计流程图

（1）定义 fire 为继电器端口 P3.7，定义 fire_key 为发射按键端口 P3.5，并初始化发射按键键值。当发射按键按下后松开，继电器导通，延时 1 秒，继电器断开，然后置密码标志位并刷新显示系统界面。相应程序如下：

```
...
sbit fire=P3^7;                    //继电器端口 P3.7 定义
sbit fire_key=P3^5;                //发射按键端口 P3.5 定义
bit fire_last=1,fire_now=1;        //发射按键键值初始化

void main()                        //主函数
{
    ...
    while(1)
    {
        if(lock_flag==0)           //锁定标志位为 0
        {
            if(code_flag==0)       //密码标志位为 0
            {
```

```
            }
            else                        //密码标志位为 1
            {
                fire_last=fire_now;
                fire_now =fire_key;     //发射按键键值刷新
                if(fire_last==0&&fire_now==1)      //发射按键按下后松开
                {
                    fire=0;             //继电器导通
                    Delay1000ms();      //延时 1000ms
                    fire=1;             //继电器断开
                    code_flag=0;        //置密码标志位为 1
                    lcd_pos(2,1);
                    lcdwrite_string(input);          //显示输入界面 input
                }
            }
        }
        else                            //锁定标志位为 1
        {
        }
    }
}
```

（2）定义所需的显示内容为 code 型字符串数组，定义 set_key 为重置按键端口 P1.7，并初始化重置按键键值，定义 set_flag 为重置标志位。当重置按键按下后松开，取反重置标志位以切换系统状态：当重置标志位为 1 时进入密码重置状态，依次调用按键检测函数和密码重置函数，通过数组赋值更新密码并刷新显示对应信息；当重置标志位为 0 时退出密码重置状态回到密码输入状态，同样置密码标志位并刷新显示系统界面。此外，为了进一步优化程序设计，缩短 while(1) 循环的时间，提取能够跟随重置按键执行的液晶显示部分，在循环中仅保留按键检测和密码重置部分。相应程序如下：

```
...
uchar code set[]    ={"    set code: ___ "};          //重置界面 set 定义
uchar code save[]  ={"       SAVE OK!       "};      //显示信息 save 定义
sbit set_key=P1^7;                      //重置按键端口 P1.7 定义
bit set_last=1,set_now=1;               //重置按键键值初始化
bit set_flag;                           //重置标志位定义
void set_code();                        //密码重置函数声明

void main()                             //主函数
{
    ...
```

```
    while(1)
    {
        if(lock_flag==0)                //锁定标志位为 0
        {
            if(code_flag==0)            //密码标志位为 0
            {
            }
            else                        //密码标志位为 1
            {
                ...
                set_last=set_now;
                set_now =set_key;       //重置按键键值刷新
                if(set_last==0&&set_now==1)       //重置按键按下后松开
                {
                    set_flag=~set_flag; //重置标志位取反
                    if(set_flag==0)     //重置标志位为 0
                    {
                        code_flag=0;    //置密码标志位为 0
                        lcd_pos(2,1);
                        lcdwrite_string(input);       //显示输入界面 input
                    }
                    else                //重置标志位为 1
                    {
                        lcd_pos(2,1);
                        lcdwrite_string(set);         //显示重置界面 set
                    }
                }
                if(set_flag==1)         //重置标志位为 1
                {
                    scan_key();         //按键检测
                    set_code();         //密码重置
                }
            }
        }
        else                            //锁定标志位为 1
        {
        }
    }
}

void set_code()                         //密码重置函数
{
    if(count==3)                        //输入次数为 3
    {
```

```
            count=0;                    //输入次数清零
            code0[0]=code1[0];
            code0[1]=code1[1];
            code0[2]=code1[2];          //正确密码赋值
            lcd_pos(2,1);
            lcdwrite_string(save);      //显示信息 save
        }
    }
```

4.2.4 step4：系统锁定模块设计

本步骤主要完成的具体任务是在系统锁定状态下能够通过同时按下重置按键和 1 数字键解锁，回到密码输入界面，程序设计流程图如图 3.7 所示。

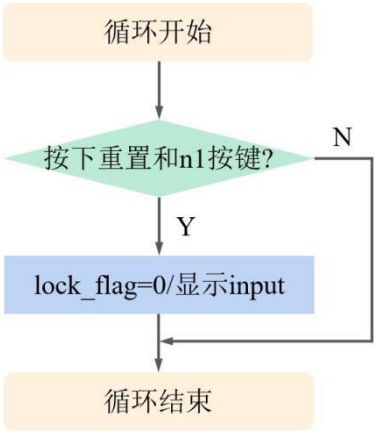


图 3.7 系统锁定模块程序设计流程图

针对本步骤的设计任务，检测重置按键和 1 数字键是否按下并置锁定标志位即可，同样置标志位切换状态时刷新显示系统界面，下一次 while(1)循环即回到密码输入状态。相应程序如下：

```
    void main()                     //主函数
    {
        ...
        while(1)
        {
            if(lock_flag==0)        //锁定标志位为 0
            {
                if(code_flag==0)    //密码标志位为 0
                {
                }
                else                //密码标志位为 1
                {
                }
```

```
        }
        else                            //锁定标志位为 1
        {
            if(set_key==0&&n1==0)   //重置按键和 1 数字键同时按下
            {
                lock_flag=0;            //置锁定标志位为 0
                lcd_pos(2,1);
                lcdwrite_string(input); //显示输入界面 input
            }
        }
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"1602LCD.h"
#define uchar unsigned char
uchar code disp1[]={"Ke Ji Qiang Guo!"};          //宣传标语 disp1 定义
uchar code disp2[]={"Shi Gan Bao Guo!"};          //宣传标语 disp2 定义
uchar code name[]={"      DONGFENG      "};   //系统名称 name 定义
uchar code input[]={"input code: ___ "};          //输入界面 input 定义
uchar code set[]   ={"   set code: ___ "};          //重置界面 set 定义
uchar code ready[]={"        READY!        "};    //显示信息 ready 定义
uchar code lock[]  ={"        LOCK!!        "};    //显示信息 lock 定义
uchar code save[]  ={"      SAVE OK!      "};    //显示信息 save 定义
sbit n1=P1^4;                           //1 数字键端口 P1.4 定义
bit n1_last=1,n1_now=1;                 //1 数字键键值初始化
sbit n2=P1^5;                           //2 数字键端口 P1.5 定义
bit n2_last=1,n2_now=1;                 //2 数字键键值初始化
sbit n3=P1^6;                           //3 数字键端口 P1.6 定义
bit n3_last=1,n3_now=1;                 //3 数字键键值初始化
sbit n4=P3^2;                           //4 数字键端口 P3.2 定义
bit n4_last=1,n4_now=1;                 //4 数字键键值初始化
sbit n5=P3^3;                           //5 数字键端口 P3.3 定义
bit n5_last=1,n5_now=1;                 //5 数字键键值初始化
sbit n6=P3^4;                           //6 数字键端口 P3.4 定义
bit n6_last=1,n6_now=1;                 //6 数字键键值初始化
sbit fire=P3^7;                         //继电器端口 P3.7 定义
sbit fire_key=P3^5;                     //发射按键端口 P3.5 定义
bit fire_last=1,fire_now=1;             //发射按键键值初始化
```

```
sbit set_key=P1^7;                  //重置按键端口 P1.7 定义
bit set_last=1,set_now=1;           //重置按键键值初始化
bit lock_flag;                      //锁定标志位定义
bit code_flag;                      //密码标志位定义
bit set_flag;                       //重置标志位定义
uchar code0[]={1,2,3};              //正确密码数组定义
uchar code1[]={0,0,0};              //密码输入数组定义
uchar count;                        //密码输入次数定义
uchar error_count;                  //密码错误次数定义
void Delay1000ms();                 //延时 1000ms 函数声明
void scan_key();                    //按键检测函数声明
void input_code();                  //密码输入函数声明
void set_code();                    //密码重置函数声明

void main()                         //主函数
{
    lcd_init();                     //液晶显示初始化
    lcd_pos(1,1);                   //显示位置第 1 行第 1 列
    lcdwrite_string(disp1);         //显示宣传标语 disp1
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
    lcdwrite_string(disp2);         //显示宣传标语 disp2
    Delay1000ms();                  //延时 1000ms
    lcd_clear();                    //液晶显示清屏
    lcd_pos(1,1);                   //显示位置第 1 行第 1 列
    lcdwrite_string(name);          //显示系统名称 name
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
    lcdwrite_string(input);         //显示输入界面 input
    while(1)
    {
        if(lock_flag==0)            //锁定标志位为 0
        {
            if(code_flag==0)        //密码标志位为 0
            {
                scan_key();         //按键检测
                input_code();       //密码输入
            }
            else                    //密码标志位为 1
            {
                fire_last=fire_now;
                fire_now =fire_key;     //发射按键键值刷新
                set_last=set_now;
                set_now =set_key;       //重置按键键值刷新
                if(fire_last==0&&fire_now==1)      //发射按键按下后松开
                {
```

```
                    fire=0;                 //继电器导通
                    Delay1000ms();          //延时 1000ms
                    fire=1;                 //继电器断开
                    code_flag=0;            //置密码标志位为 1
                    lcd_pos(2,1);
                    lcdwrite_string(input);             //显示输入界面 input
                }
                if(set_last==0&&set_now==1)         //重置按键按下后松开
                {
                    set_flag=~set_flag; //重置标志位取反
                    if(set_flag==0)       //重置标志位为 0
                    {
                        code_flag=0;    //置密码标志位为 0
                        lcd_pos(2,1);
                        lcdwrite_string(input);         //显示输入界面 input
                    }
                    else                    //重置标志位为 1
                    {
                        lcd_pos(2,1);
                        lcdwrite_string(set);           //显示重置界面 set
                    }
                }
                if(set_flag==1)           //重置标志位为 1
                {
                    scan_key();           //按键检测
                    set_code();           //密码重置
                }
            }
        }
        else                              //锁定标志位为 1
        {
            if(set_key==0&&n1==0)   //重置按键和 1 数字键同时按下
            {
                lock_flag=0;              //置锁定标志位为 0
                lcd_pos(2,1);
                lcdwrite_string(input); //显示输入界面 input
            }
        }
    }
}

void scan_key()                           //按键检测函数
{
    n1_last=n1_now;
```

```
n1_now=n1;                          //1 数字键键值刷新
n2_last=n2_now;
n2_now=n2;                          //2 数字键键值刷新
n3_last=n3_now;
n3_now=n3;                          //3 数字键键值刷新
n4_last=n4_now;
n4_now=n4;                          //4 数字键键值刷新
n5_last=n5_now;
n5_now=n5;                          //5 数字键键值刷新
n6_last=n6_now;
n6_now=n6;                          //6 数字键键值刷新
if(n1_last==0&&n1_now==1)                   //1 数字键按下后松开
{
    code1[count]=1;                 //保存数字 1
    lcd_pos(2,13+count);
    lcdwrite_sz(1);                 //显示数字 1
    count++;                        //输入次数计数
}
else if(n2_last==0&&n2_now==1)              //2 数字键按下后松开
{
    code1[count]=2;                 //保存数字 2
    lcd_pos(2,13+count);
    lcdwrite_sz(2);                 //显示数字 2
    count++;                        //输入次数计数
}
else if(n3_last==0&&n3_now==1)              //3 数字键按下后松开
{
    code1[count]=3;                 //保存数字 3
    lcd_pos(2,13+count);
    lcdwrite_sz(3);                 //显示数字 3
    count++;                        //输入次数计数
}
else if(n4_last==0&&n4_now==1)              //4 数字键按下后松开
{
    code1[count]=4;                 //保存数字 4
    lcd_pos(2,13+count);
    lcdwrite_sz(4);                 //显示数字 4
    count++;                        //输入次数计数
}
else if(n5_last==0&&n5_now==1)              //5 数字键按下后松开
{
    code1[count]=5;                 //保存数字 5
    lcd_pos(2,13+count);
    lcdwrite_sz(5);                 //显示数字 5
```

```
        count++;                        //输入次数计数
    }
    else if(n6_last==0&&n6_now==1)              //6 数字键按下后松开
    {
        code1[count]=6;                 //保存数字 6
        lcd_pos(2,13+count);
        lcdwrite_sz(6);                 //显示数字 6
        count++;                        //输入次数计数
    }
}

void input_code()                       //密码输入函数
{
    if(count==3)                        //输入次数为 3
    {
        count=0;                        //输入次数清零
        if(code1[0]==code0[0]&&code1[1]==code0[1]&&code1[2]==code0[2])
                                        //密码正确
        {
            error_count=0;              //错误次数清零
            code_flag=1;                //置密码标志位为 1
            lcd_pos(2,1);
            lcdwrite_string(ready);     //显示信息 ready
        }
        else                            //密码错误
        {
            error_count++;              //错误次数计数
            if(error_count==3)          //错误次数为 3
            {
                error_count=0;          //错误次数清零
                lock_flag=1;            //置锁定标志位为 1
                lcd_pos(2,1);
                lcdwrite_string(lock);  //显示信息 lock
            }
            else                        //错误次数为 1/2
            {
                lcd_pos(2,1);
                lcdwrite_string(input); //显示输入界面 input
            }
        }
    }
}

void set_code()                         //密码重置函数
```

```
{
    if(count==3)                    //输入次数为 3
    {
        count=0;                    //输入次数清零
        code0[0]=code1[0];
        code0[1]=code1[1];
        code0[2]=code1[2];          //正确密码赋值
        lcd_pos(2,1);
        lcdwrite_string(save);      //显示信息 save
    }
}

void Delay1000ms()                  //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

5.本章小结

本章设计开发了一种导弹发射控制系统的密码模块，主要涉及按键输入和显示输出的同步控制，通过调用相关液晶显示函数实现控制系统密码输入、导弹发射、密码重置和状态切换时系统界面的刷新显示。使用液晶显示存在一些注意事项和使用技巧，总结如下：（1）通常情况下刷新显示直接对原有内容进行部分覆盖即可，为了避免前后内容长度不一致所造成的覆盖不完全的情况，可以在字符串定义时即规划字符串长度，将同一位置的显示内容定义为同一长度；（2）为了缩短 while(1)循环的时间，同时避免重复的界面刷新，可以将液晶显示尽量跟随按键执行，在循环中仅保留实时刷新的部分，根据程序设计的具体需求安排液晶显示的调用位置。同时，本章展示了层级结构控制系统的程序设计方法，延续第

二章的程序设计思路进一步深化了标志位的使用，即控制系统切换状态仅改变标志位数值，再根据标志位数值执行相应模块程序，并以此为依据规划 step by step 程序设计步骤。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用液晶显示的控制功能，并能够独立完成层级结构控制系统的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，完善导弹发射控制系统的后续功能，实现直流电机、独立按键、液晶显示的联动控制，例如使用直流电机调整发射方位，具体功能自拟。

一种单轴运动工作台控制系统开发

1.学习目标

1. 能根据控制系统任务要求完成直流电机、独立按键、液晶显示的综合应用；
2. 能应用外部中断实现需要快速响应和及时处理的控制功能；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

中断系统是单片机的重要组成部分，其中断响应和处理过程如图 4.1 所示。当中断源发出的中断请求被允许时，单片机会暂时中止执行当前程序，转而执行中断服务程序处理中断服务请求，处理结束后再回到原来中止程序位置，继续执行被中断的程序。由于中断系统工作方式的实时特性，常用于实现实时控制、故障处理、紧急停止等需要快速响应和及时处理的控制功能，大大提高了单片机的工作效率。

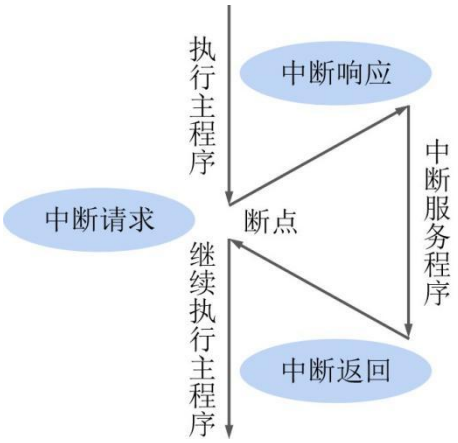


图 4.1 中断响应和处理过程示意图

AT89C52 单片机的中断系统共有 5 个中断源，分别为外部中断 0、外部中断 1、定时器 0、定时器 1、串口通信，其中外部中断 0 和外部中断 1 是由外部信号产生的中断请求。本章提出一例“设计一种单轴运动工作台控制系统”的课题需求，学习外部中断的初始设置和使用方法，并掌握单片机综合控制系统的程序设计方法。

任务题目：设计一种单轴运动工作台控制系统

功能要求：该系统能够通过按下不同按键，控制电机旋转，带动工作台沿轴向进行往返运动。系统有两种控制模式，手动模式和自动模式，只有在电机停止的情况下可以按下模式按键在这两种模式之间切换。分别设定左移（手动模式）/启动（自动模式）按键、右移（手动模式）/停止（自动模式）按键、模式按键、限位信号、急停按键、解除按键，系统独立按键功能配置如表 4.1 所示。

表 4.1 系统独立按键功能配置表

左移（手动模式） 启动（自动模式）	解除按键 K6		右移（手动模式） 停止（自动模式）
限位信号 K1	急停按键 K2		模式按键 K4

具体控制要求如下：

1.初始界面显示。系统上电后默认进入手动模式，在 1602LCD 第 1 行显示系统名称“CNC system”（CNC：Computerized Numerical Control 数控机床），第 2 行显示手动模式界面，依次显示系统模式、电机状态、运行情况信息“manu stop normal”。

2.手动模式控制。按下左移按键，工作台向左移动（电机正转），放开即停；按下右移按键，工作台向右移动（电机反转），放开即停。运行过程中如果遇到左右限位信号（外部中断 0 按键触发），电机立即停止，此时只能响应反向按键。液晶显示实时刷新电机状态“stop/fwd/back”和运行情况“normal/sen_l/sen_r”。

3.自动模式控制。第 2 行显示自动模式界面，依次显示系统模式、电机状态、运行次数信息“auto stop No.00”。

按下启动按键并松开后，工作台向左移动（电机正转）；按下停止按键并松开后，工作台停止移动（电机停止）。运行过程中如果遇到左右限位信号（外部中断 0 按键触发），电机反向运行。液晶显示实时刷新电机状态“stop/fwd/back”和运行次数“No.00”，其中往返运动记运行 1 次。

4.紧急停止功能。系统在任意工作状态，如果遇到突发情况，可以按下急停按键（外部中断 1 按键触发），系统立即停止，显示界面对应电机状态位置显示“SOS!”信息。当排除突发情况，可以按下解除按键，消隐“SOS!”信息，系统回到手动模式继续工作。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输入模块

单片机输入模块包括独立按键控制模块和外部信号发生模块。

（1）独立按键控制模块主要用于系统运行过程中模式切换、电机运动、解除急停各控制功能的人机操作。单片机通过判断 I/O 接口的电平状态，能够识别对应按键是否按下或松开，其控制原理已在第二章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P1.4、P1.5、P1.7、P3.5 引脚。

（2）外部信号发生模块主要用于提供限位信号和急停信号，产生中断请求触发外部中断 0 和外部中断 1。该模块的选型应根据实际控制装置的要求进行，本例不作展开。

外部中断的触发方式有电平触发方式和跳沿触发方式两种，电平触发方式为低电平信号触发中断请求，跳沿触发方式为下降沿信号触发中断请求。本例使用跳沿触发方式，接口电路如图 4.2 所示，其中外部中断 0 使用 P3.2 引脚，在开发板上接有按键 K1，外部中断 1 使用 P3.3 引脚，在开发板上接有按键 K2。以外部中断 0 为例，当 K1 按下时 P3.2 会产生下降沿信号从而触发外部中断 0。

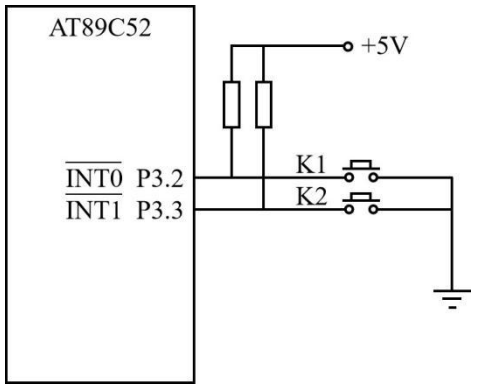


图 4.2 外部中断接口电路图

3.2 单片机输出模块

单片机输出模块包括直流电机控制模块和液晶显示控制模块。

（1）直流电机控制模块主要用于工作台往返运动的动力驱动。该功能由单片机输出信号控制电机驱动，对旋转角度的控制要求不高，可以选择直流电机及其驱动模块的控制方案。同时该模块的选型应根据实际控制装置的要求进行，本例不作展开。直流电机的控制原理已在第一章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P2.6/P2.7 引脚。

（2）液晶显示控制模块主要用于系统运行过程中各项信息数据的实时刷新显示。单片机通过输出数据和命令至 1602LCD，从而实现相应的数字、字母、符号显示，其控制原理已在第三章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P0+P2.0/P2.1/P2.2 引脚。

3.3 单片机内部配置

单片机内部配置包括外部中断配置，主要用于外部信号发生模块产生限位信号和急停信号后的中断响应和中断处理，使用外部中断 0（P3.2）和外部中断 1（P3.3）分别实现左右限位和紧急停止相关的控制功能。

系统端口及硬件配置如表 4.2 所示。

表 4.2 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P3.5	模式按键
2	P1.4	左移/启动按键
3	P1.7	右移/停止按键
4	P1.5	解除按键
5	P2.6/P2.7	直流电机
6	P0+P2.0/P2.1/P2.2	液晶显示
7	外部中断 0（P3.2）	限位信号
8	外部中断 1（P3.3）	急停按键

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“系统工作”/“系统停止”和“手动模式”/“自动模式”两组不同级别的模块，分别设定 bit 类型的急停标志位 limit_flag 和模式标志位 mode_flag，并将模式模块置于系统模块的工作状态下。系统上电后默认 limit_flag=0 进入“系统工作”模块，此时系统正常运行。通过按下模式按键取反标志位，置 mode_flag=~mode_flag，当 mode_flag=0 时进入“手动模式”模块，当 mode_flag=1 时进入“自动模式”模块。当 limit_flag=1 时进入“系统停止”模块，通过按下解除按键置 limit_flag=0 回到“系统工作”模块。

系统标志位对应关系如表 4.3 所示。

表 4.3 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
limit_flag	0	系统工作
	1	系统停止
mode_flag	0	手动模式
	1	自动模式

系统软件架构如图 4.3 所示。

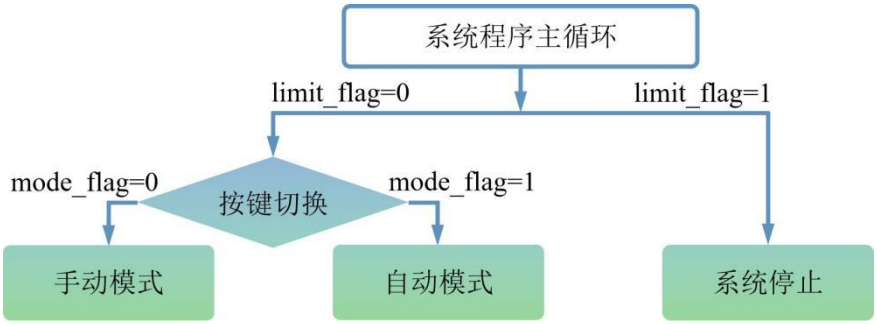


图 4.3 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由四个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 4.4 所示。

表 4.4 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	系统程序框架设计	标志位划分系统状态的应用
Step 2	手动模式模块设计	外部中断的应用
Step 3	自动模式模块设计	外部中断的应用
Step 4	系统停止模块设计	外部中断的应用

4.2.1 step1：系统程序框架设计

在进行控制系统各个模块的程序设计之前，搭建系统程序框架，即将软件架构以程序语言的形式体现。此外，系统上电后需要进行初始界面显示，放置在主函数的 while(1)循环前执行。

针对本步骤的设计任务，根据软件架构使用标志位划分系统状态以执行不同模块程序即可，同时根据功能要求调用液晶显示函数实现初始界面显示。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，其中“motor.h”为直流电机头文件，“1602LCD.h”为液晶显示头文件。定义 K4 为模式按键端口 P3.5，并初始化模式按键键值。定义 limit_flag 为急停标志位，mode_flag 为模式标志位：当急停标志位为 0 时，此时系统工作，按下模式按键取反模式标志位，根据模式标志位为 0 或 1 分别执行“手动模式”或“自动模式”模块程序；当急停标志位为 1 时，此时系统停止，执行“系统停止”模块程序。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#define uchar unsigned char
sbit K4=P3^5;                          //模式按键端口 P3.5 定义
```

```
bit K4_last=1,K4_now=1;                 //模式按键键值初始化
bit limit_flag;                         //急停标志位定义
bit mode_flag;                          //模式标志位定义

void main()                             //主函数
{
    while(1)
    {
        if(limit_flag==0)               //急停标志位为 0
        {
            K4_last=K4_now;
            K4_now =K4;                 //模式按键键值刷新
            if(K4_last==0&&K4_now==1)           //模式按键按下后松开
            {
                mode_flag=~mode_flag;           //模式标志位取反
            }
            if(mode_flag==0)            //模式标志位为 0
            {
            }
            else                        //模式标志位为 1
            {
            }
        }
        else                            //急停标志位为 1
        {
        }
    }
}
```

（2）定义初始界面显示所需的系统名称、系统模式、电机状态、运行情况为 code 型字符串数组：进行液晶显示初始化，指定第 1 行第 1 列显示系统名称 name，第 2 行第 1 列显示手动模式界面，依次显示系统模式 manu、电机状态 stop、运行情况 norm。相应程序如下：

```
...
uchar code name[] ={"     CNC system     "};       //系统名称 name 定义
uchar code disp_manu[]  ={"manu "};                 //系统模式 manu 定义
uchar code disp_stop[]    ={"stop "};               //电机状态 stop 定义
uchar code disp_norm[]  ={"normal"};                //运行情况 norm 定义

void main()                             //主函数
{
    lcd_init();                         //液晶显示初始化
    lcd_pos(1,1);                       //显示位置第 1 行第 1 列
```

```
    lcdwrite_string(name);              //显示系统名称 name
    lcd_pos(2,1);                       //显示位置第 2 行第 1 列
    lcdwrite_string(disp_manu);         //显示系统模式 manu
    lcdwrite_string(disp_stop);         //显示电机状态 stop
    lcdwrite_string(disp_norm);         //显示运行情况 norm
    while(1)
    {
    }
}
```

4.2.2 step2：手动模式模块设计

本步骤主要完成的具体任务是按下左移按键，电机正转，放开即停，按下右移按键，电机反转，放开即停。运行过程中如果遇到限位信号，电机停止，只能响应反向按键。同时，手动模式界面实时刷新电机状态和运行情况。

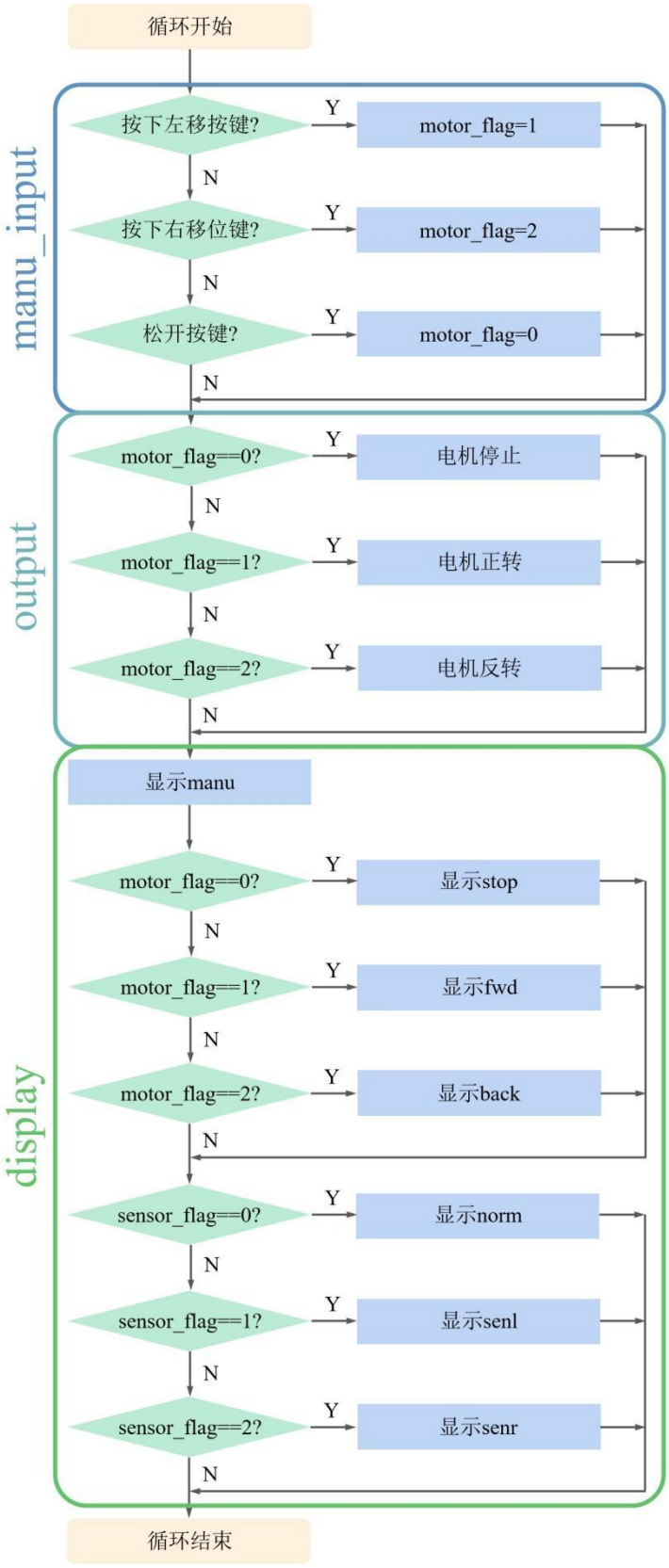
首先需要明确系统主程序和中断程序分别实现的控制功能。其中系统主程序需要实现按键输入、电机输出、液晶显示相关的控制功能，外部中断 0 需要实现左右限位相关的控制功能。需要注意的是，在不考虑中断优先级的情况下，系统执行中断程序时无法再响应其他的命令请求，因此中断程序应尽可能地快速执行以便回到主程序继续响应其他请求。基于这点考虑，可以在中断程序中根据功能要求仅改变标志位数值，回到主程序再根据标志位数值执行电机输出和液晶显示。

表 4.5 系统标志位对应关系表

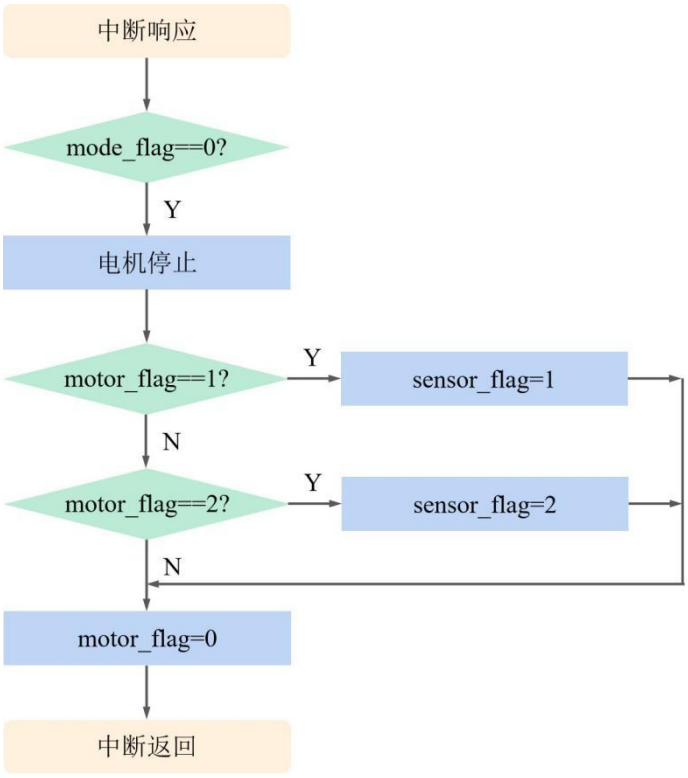
标志位名称	标志位数值	对应当前状态
motor_flag	0	电机停止
	1	电机正转
	2	电机反转
sensor_flag	0	正常运行
	1	遇到左限位
	2	遇到右限位

根据系统主程序实现的控制功能，设定 unsigned char 类型的电机标志位 motor_flag，当 motor_flag=0 时为电机停止状态，当 motor_flag=1 时为电机正转状态，当 motor_flag=2 时为电机反转状态。同时，根据外部中断 0 实现的控制功能，设定 unsigned char 类型的限位标志位 sensor_flag，当 sensor_flag=0 时为正

常运行状态，当 sensor_flag=1 时为遇到左限位状态，当 sensor_flag=2 时为遇到右限位状态。系统标志位对应关系如表 4.5 所示。



(a)



(b)

图 4.4 手动模式模块程序设计流程图

然后使用电机标志位和限位标志位再次梳理手动模式的具体功能。系统主程序的程序设计流程图如图 4.4(a)所示，分为手动输入部分（manu_input）、电机输出部分（output）和液晶显示部分（display），其中遇到限位信号只能响应反向按键的功能实现可以在按键检测时通过限位标志位进行限制。外部中断 0 的程序设计流程图如图 4.4(b)所示，左右限位信号可以通过当前电机状态判断，电机正转意味着工作台左移，此时遇到的限位信号为左限位信号，电机反转意味着工作台右移，此时遇到的限位信号为右限位信号。

针对本步骤的设计任务，设计思路应首先考虑实现系统主程序部分，检测按键是否按下并置电机标志位，根据电机标志位控制电机正反停转，同时显示手动模式并根据当前标志位显示电机状态和运行情况，然后在此基础上实现外部中断 0 部分，进行中断初始化配置，建立中断服务程序，并置电机标志位和限位标志位。

（1）在手动模式下，定义 K5 为左移按键端口 P1.4，K8 为右移按键端口 P1.7，并初始化左移、右移按键键值，定义 motor_flag 为电机标志位，sensor_flag 为限位标志位，根据手动模式模块程序设计流程图建立手动输入函数，调用“motor.h”

头文件中的正转、反转、停止函数建立电机输出函数，并声明程序中使用的所有函数。相应程序如下：

```
...
sbit K5=P1^4;                          //左移按键端口 P1.4 定义
bit K5_last=1,K5_now=1;                //左移按键键值初始化
sbit K8=P1^7;                          //右移按键端口 P1.7 定义
bit K8_last=1,K8_now=1;                //右移按键键值初始化
uchar motor_flag;                      //电机标志位定义
uchar sensor_flag;                     //限位标志位定义
void manu_input();                     //手动输入函数声明
void output();                         //电机输出函数声明

void manu_input()                      //手动输入函数
{
    if(K5==0&&sensor_flag!=1)          //正常运行/右限位状态下按下左移按键
    {
        motor_flag=1;                  //置电机标志位为 1
        sensor_flag=0;                 //置限位标志位为 0
    }
    else if(K8==0&&sensor_flag!=2)     //正常运行/左限位状态下按下右移按键
    {
        motor_flag=2;                  //置电机标志位为 2
        sensor_flag=0;                 //置限位标志位为 0
    }
    else                               //左移/右移按键松开
    {
        motor_flag=0;                  //置电机标志位为 0
    }
}

void output()                          //电机输出函数
{
    if(motor_flag==0)                  //电机标志位为 0
    {
        stop();                        //电机停止
    }
    else if(motor_flag==1)             //电机标志位为 1
    {
        fwd();                         //电机正转
    }
    else if(motor_flag==2)             //电机标志位为 2
    {
        back();                        //电机反转
```

```
    }
}
```

（2）定义所需的显示内容为 code 型字符串数组，其中同一位置的显示内容为同一长度，根据手动模式模块程序设计流程图建立液晶显示函数。同时，在主函数的 while(1)循环中依次调用手动输入函数、电机输出函数和液晶显示函数。相应程序如下：

```
...
uchar code disp_fwd[]    ={"fwd    "};              //电机状态 fwd 定义
uchar code disp_back[]   ={"back "};                //电机状态 back 定义
uchar code disp_senl[]   ={"sen_l "};               //运行情况 senl 定义
uchar code disp_senr[]   ={"sen_r "};               //运行情况 senr 定义
void display();                             //液晶显示函数声明

void main()                                 //主函数
{
    ...
    while(1)
    {
        if(limit_flag==0)                   //急停标志位为 0
        {
            K4_last=K4_now;
            K4_now =K4;                     //模式按键键值刷新
            if(K4_last==0&&K4_now==1&&motor_flag==0)
                                            //电机停止状态下模式按键按下后松开
            {
                mode_flag=~mode_flag;               //模式标志位取反
            }
            if(mode_flag==0)            //模式标志位为 0
            {
                manu_input();           //手动输入
                output();               //电机输出
                display();              //液晶显示
            }
            else                        //模式标志位为 1
            {
            }
        }
        else                            //急停标志位为 1
        {
        }
    }
}
```

```
void display()                          //液晶显示函数
{
    lcd_pos(2,1);
    lcdwrite_string(disp_manu);         //显示系统模式 manu
    if(motor_flag==0)                   //电机标志位为 0
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_stop);     //显示电机状态 stop
    }
    else if(motor_flag==1)              //电机标志位为 1
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_fwd);      //显示电机状态 fwd
    }
    else if(motor_flag==2)              //电机标志位为 2
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_back);     //显示电机状态 back
    }
    if(sensor_flag==0)                  //限位标志位为 0
    {
        lcd_pos(2,11);
        lcdwrite_string(disp_norm);     //显示运行情况 norm
    }
    else if(sensor_flag==1)             //限位标志位为 1
    {
        lcd_pos(2,11);
        lcdwrite_string(disp_senl);     //显示运行情况 senl
    }
    else if(sensor_flag==2)             //限位标志位为 2
    {
        lcd_pos(2,11);
        lcdwrite_string(disp_senr);     //显示运行情况 senr
    }
}
```

（3）在此基础上继续完成外部中断 0 部分。

首先，一个中断源发出的中断请求被允许需要打开总中断允许开关和该中断源的中断允许开关，其中总中断允许开关为 EA=1，外部中断 0 开关为 EX0=1，外部中断 1 开关为 EX1=1。此外，外部中断还需要通过标志位 IT0 和 IT1 选择中断触发方式，以外部中断 0 为例，当 IT0=0 时为电平触发方式，当 IT0=1 时为跳

沿触发方式。以上这些设置统称为中断初始化配置，放置在主函数的 while(1)循环前执行。

进行中断初始化配置之后，此时当外部中断发出中断请求时，单片机将会响应该中断请求并自动跳转执行中断服务程序。中断服务程序不需要进行声明，也不需要进行调用，其建立的一般形式为：

void 函数名() interrupt n

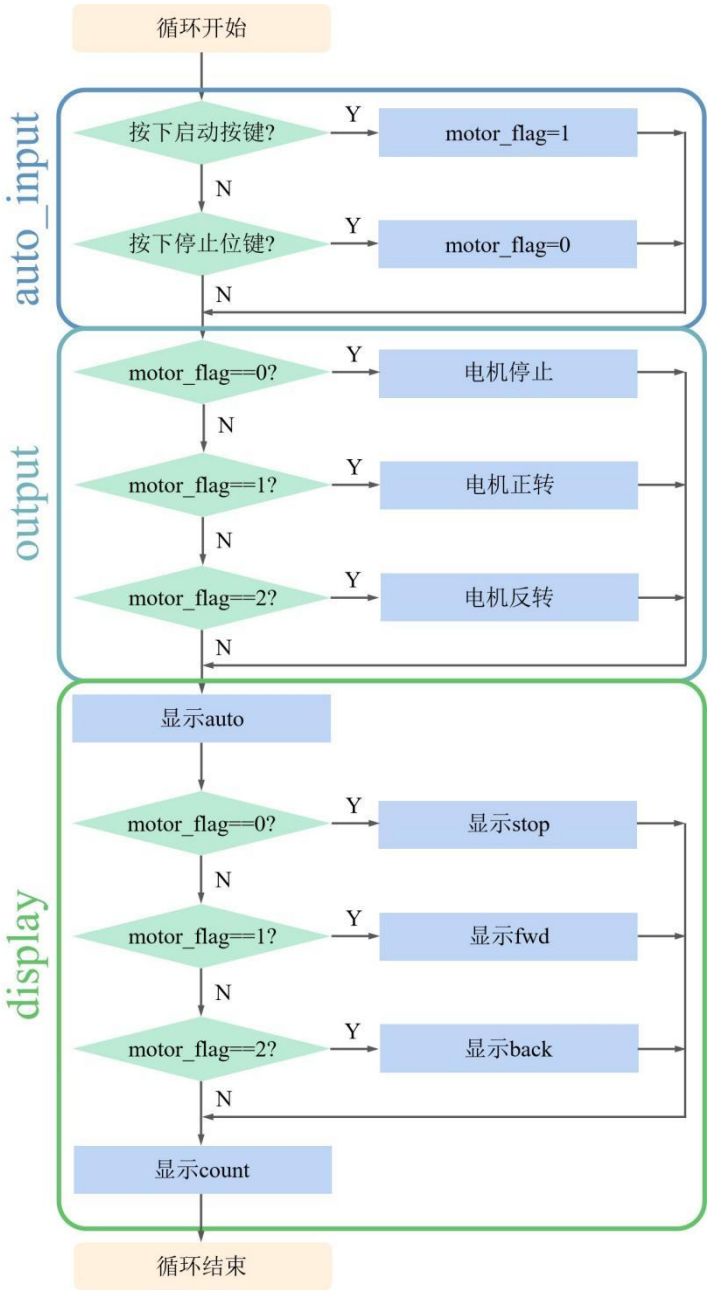
其中 n 为中断号，用于识别不同的中断源并指向对应的中断地址。外部中断 0 的中断号为 0，外部中断 1 的中断号为 2，以外部中断 0 为例，可以建立中断服务程序为 void int_ex0() interrupt 0。

根据手动模式模块程序设计流程图进行中断初始化配置并建立外部中断 0 函数。相应程序如下：

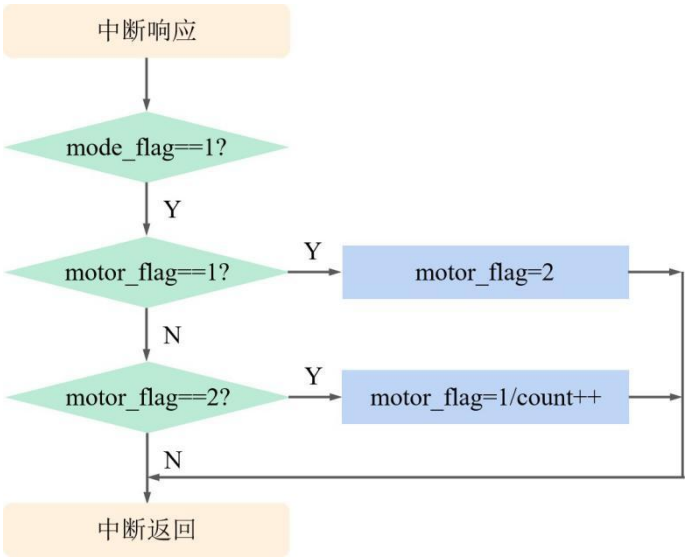
```
void main()                          //主函数
{
    EA=1;                            //打开总中断允许开关
    EX0=1;                           //打开外部中断 0 开关
    IT0=1;                           //选择跳沿触发方式
    while(1)
    {
    }
}

void int_ex0() interrupt 0           //外部中断 0 函数
{
    if(mode_flag==0)                 //模式标志位为 0
    {
        stop();                      //电机停止
        if(motor_flag==1)            //电机标志位为 1
        {
            sensor_flag=1;           //置限位标志位为 1
        }
        else if(motor_flag==2)       //电机标志位为 2
        {
            sensor_flag=2;           //置限位标志位为 2
        }
        motor_flag=0;                //置电机标志位为 0
    }
}
```

4.2.3 step3：自动模式模块设计

本步骤主要完成的具体任务是按下启动按键并松开后，电机正转，按下停止按键并松开后，电机停止。运行过程中如果遇到限位信号，电机反转。同时，自动模式界面实时刷新电机状态和运行次数。与手动模式模块设计相同，系统主程序需要实现按键输入、电机输出、液晶显示相关的控制功能，程序设计流程图如图 4.5(a)所示，外部中断 0 需要实现左右限位相关的控制功能，程序设计流程图如图 4.5(b)所示，其中 count 为运行次数。



(a)



(b)

图 4.5 自动模式模块程序设计流程图

针对本步骤的设计任务，设计思路同样应首先考虑实现系统主程序部分，检测按键是否按下并置电机标志位，根据电机标志位控制电机正反停转，同时显示自动模式和运行次数并根据电机标志位显示电机状态，然后在此基础上实现外部中断 0 部分，通过当前电机状态判断左右限位信号，置反向电机标志位并对运行次数计数。此外，自动模式和手动模式都需要根据电机标志位控制电机转停并显示电机状态，可以共用电机输出函数和液晶显示函数。

（1）在自动模式下，定义 K5 为启动按键端口 P1.4，K8 为停止按键端口 P1.7，并初始化启动、停止按键键值，根据自动模式模块程序设计流程图建立自动输入函数。相应程序如下：

```
...
sbit K5=P1^4;                    //启动按键端口 P1.4 定义
bit K5_last=1,K5_now=1;          //启动按键键值初始化
sbit K8=P1^7;                    //停止按键端口 P1.7 定义
bit K8_last=1,K8_now=1;          //停止按键键值初始化
void auto_input();               //自动输入函数声明

void auto_input()                //自动输入函数
{
    K5_last=K5_now;
    K5_now=K5;                   //启动按键键值刷新
    K8_last=K8_now;
    K8_now=K8;                   //停止按键键值刷新
    if(K5_last==0&&K5_now==1)                //启动按键按下后松开
```

```
    {
        motor_flag=1;                   //置电机标志位为 1
    }
    else if(K8_last==0&&K8_now==1)          //停止按键按下后松开
    {
        motor_flag=0;                   //置电机标志位为 0
    }
}
```

（2）定义所需的显示内容为 code 型字符串数组，定义 count 为运行次数，根据自动模式模块程序设计流程图完善液晶显示函数。同时，在主函数的 while(1)循环中依次调用自动输入函数、电机输出函数和液晶显示函数。相应程序如下：

```
...
uchar code disp_auto[]   ={"auto "};            //系统模式 auto 定义
uchar code disp_no[]     ={" No."};             //运行次数 no 定义
uchar count;                            //运行次数定义

void main()                             //主函数
{
    ...
    while(1)
    {
        if(limit_flag==0)               //急停标志位为 0
        {
            K4_last=K4_now;
            K4_now =K4;                 //模式按键键值刷新
            if(K4_last==0&&K4_now==1&&motor_flag==0)
                                        //电机停止状态下模式按键按下后松开
            {
                mode_flag=~mode_flag;           //模式标志位取反
            }
            if(mode_flag==0)            //模式标志位为 0
            {
                manu_input();           //手动输入
            }
            else                        //模式标志位为 1
            {
                auto_input();           //自动输入
            }
            output();                   //电机输出
            display();                  //液晶显示
        }
        else                            //急停标志位为 1
        {
```

```
        }
    }
}

void display()                          //液晶显示函数
{
    if(motor_flag==0)                   //电机标志位为 0
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_stop);     //显示电机状态 stop
    }
    else if(motor_flag==1)              //电机标志位为 1
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_fwd);      //显示电机状态 fwd
    }
    else if(motor_flag==2)              //电机标志位为 2
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_back);     //显示电机状态 back
    }
    if(mode_flag==0)                    //模式标志位为 0
    {
        lcd_pos(2,1);
        lcdwrite_string(disp_manu);     //显示系统模式 manu
        if(sensor_flag==0)              //限位标志位为 0
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_norm);//显示运行情况 norm
        }
        else if(sensor_flag==1)         //限位标志位为 1
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_senl); //显示运行情况 senl
        }
        else if(sensor_flag==2)         //限位标志位为 2
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_senr); //显示运行情况 senr
        }
    }
    else                                //模式标志位为 1
    {
        lcd_pos(2,1);
```

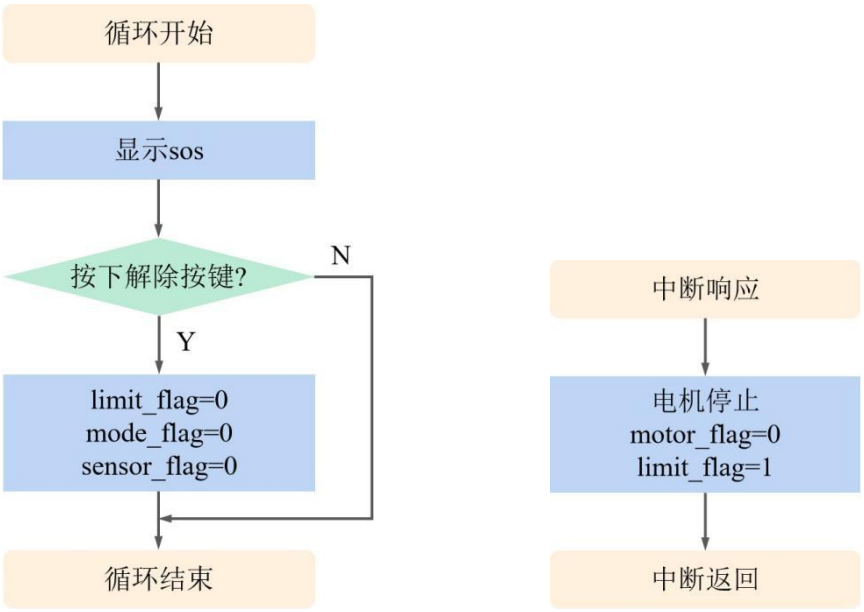
```
            lcdwrite_string(disp_auto);     //显示系统模式 auto
            lcd_pos(2,11);
            lcdwrite_string(disp_no);
            lcdwrite_sz(count/10);
            lcdwrite_sz(count%10);          //显示运行次数 count
        }
    }
```

（3）在此基础上继续完成外部中断 0 部分，根据自动模式模块程序设计流程图完善外部中断 0 函数。相应程序如下：

```
    void int_ex0() interrupt 0              //外部中断 0 函数
    {
        ...
        if(mode_flag==1)                    //模式标志位为 1
        {
            if(motor_flag==1)               //电机标志位为 1
            {
                motor_flag=2;               //置电机标志位为 2
            }
            else if(motor_flag==2)          //电机标志位为 2
            {
                motor_flag=1;               //置电机标志位为 1
                count++;                    //运行次数计数
            }
        }
    }
```

4.2.4 step4：系统停止模块设计

本步骤主要完成的具体任务是按下急停按键，系统立即停止并显示“SOS!”信息，按下解除按键，系统继续工作并消隐“SOS!”信息。其中系统主程序需要实现解除按键相关的控制功能，程序设计流程图如图 4.6(a)所示，外部中断 1 需要实现急停按键相关的控制功能，程序设计流程图如图 4.6(b)所示。



(a) (b)

图 4.6 系统停止模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现外部中断 1 部分，进行中断初始化配置，建立中断服务程序，并置电机标志位和急停标志位，然后在此基础上实现系统主程序部分，根据急停标志位刷新显示对应信息，同时检测解除按键是否按下并置各个标志位。

（1）进行中断初始化配置，打开外部中断 1 开关并选择跳沿触发方式，根据系统停止模块程序设计流程图建立外部中断 1 函数。相应程序如下：

```
void main()                          //主函数
{
    EA=1;                            //打开总中断允许开关
    EX1=1;                           //打开外部中断 1 开关
    IT1=1;                           //选择跳沿触发方式
    while(1)
    {
    }
}

void int_ex1() interrupt 2           //外部中断 1 函数
{
    stop();                          //电机停止
    motor_flag=0;                    //置电机标志位为 0
    limit_flag=1;                    //置急停标志位为 1
}
```

（2）在此基础上继续完成系统主程序部分，定义所需的显示内容为 code 型

字符串数组，定义 K6 为解除按键端口 P1.5，并初始化解除按键键值。当解除按键按下后松开，置急停标志位为 0 回到系统工作状态，置模式标志位为 0 回到手动模式状态，置限位标志位位 0 回到正常运行状态。相应程序如下：

```
...
uchar code disp_sos[]     ={"SOS! "};                //显示信息 sos 定义
sbit K6=P1^5;                                 //解除按键端口 P1.5 定义
bit K6_last=1,K6_now=1;                       //解除按键键值初始化

void main()                                   //主函数
{
    ...
    while(1)
    {
        if(limit_flag==0)                     //急停标志位为 0
        {
        }
        else                                  //急停标志位为 1
        {
            lcd_pos(2,6);
            lcdwrite_string(disp_sos);  //显示信息 sos
            K6_last=K6_now;
            K6_now =K6;                       //解除按键键值刷新
            if(K6_last==0&&K6_now==1)         //解除按键按下后松开
            {
                limit_flag=0;                 //置急停标志位为 0
                mode_flag=0;                  //置模式标志位为 0
                sensor_flag=0;                //置限位标志位为 0
            }
        }
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#define uchar unsigned char
uchar code name[] ={"      CNC system      "};        //系统名称 name 定义
uchar code disp_manu[]  ={"manu "};                   //系统模式 manu 定义
```

```
uchar code disp_auto[]   ={"auto "};              //系统模式 auto 定义
uchar code disp_stop[]   ={"stop "};              //电机状态 stop 定义
uchar code disp_fwd[]    ={"fwd   "};             //电机状态 fwd 定义
uchar code disp_back[]   ={"back "};              //电机状态 back 定义
uchar code disp_sos[]    ={"SOS! "};              //显示信息 sos 定义
uchar code disp_norm[]   ={"normal"};             //运行情况 norm 定义
uchar code disp_senl[]   ={"sen_l "};             //运行情况 senl 定义
uchar code disp_senr[]   ={"sen_r "};             //运行情况 senr 定义
uchar code disp_no[]     ={" No."};               //运行次数 no 定义
sbit K4=P3^5;                          //模式按键端口 P3.5 定义
bit K4_last=1,K4_now=1;                //模式按键键值初始化
sbit K5=P1^4;                          //左移/启动按键端口 P1.4 定义
bit K5_last=1,K5_now=1;                //左移/启动按键键值初始化
sbit K6=P1^5;                          //解除按键端口 P1.5 定义
bit K6_last=1,K6_now=1;                //解除按键键值初始化
sbit K8=P1^7;                          //右移/停止按键端口 P1.7 定义
bit K8_last=1,K8_now=1;                //右移/停止按键键值初始化
bit limit_flag;                        //急停标志位定义
bit mode_flag;                         //模式标志位定义
uchar motor_flag;                      //电机标志位定义
uchar sensor_flag;                     //限位标志位定义
uchar count;                           //运行次数定义
void manu_input();                     //手动输入函数声明
void auto_input();                     //自动输入函数声明
void output();                         //电机输出函数声明
void display();                        //液晶显示函数声明

void main()                            //主函数
{
    EA=1;                              //打开总中断允许开关
    EX0=1;                             //打开外部中断 0 开关
    IT0=1;                             //选择跳沿触发方式
    EX1=1;                             //打开外部中断 1 开关
    IT1=1;                             //选择跳沿触发方式
    lcd_init();                        //液晶显示初始化
    lcd_pos(1,1);                      //显示位置第 1 行第 1 列
    lcdwrite_string(name);             //显示系统名称 name
    lcd_pos(2,1);                      //显示位置第 2 行第 1 列
    lcdwrite_string(disp_manu);        //显示系统模式 manu
    lcdwrite_string(disp_stop);        //显示电机状态 stop
    lcdwrite_string(disp_norm);        //显示运行情况 norm
    while(1)
    {
        if(limit_flag==0)              //急停标志位为 0
```

```
        {
            K4_last=K4_now;
            K4_now =K4;                 //模式按键键值刷新
            if(K4_last==0&&K4_now==1&&motor_flag==0)
                                        //电机停止状态下模式按键按下后松开
            {
                mode_flag=~mode_flag;           //模式标志位取反
            }
            if(mode_flag==0)            //模式标志位为 0
            {
                manu_input();           //手动输入
            }
            else                        //模式标志位为 1
            {
                auto_input();           //自动输入
            }
            output();                   //电机输出
            display();                  //液晶显示
        }
        else                            //急停标志位为 1
        {
            lcd_pos(2,6);
            lcdwrite_string(disp_sos);  //显示信息 sos
            K6_last=K6_now;
            K6_now =K6;                 //解除按键键值刷新
            if(K6_last==0&&K6_now==1)           //解除按键按下后松开
            {
                limit_flag=0;           //置急停标志位为 0
                mode_flag=0;            //置模式标志位为 0
                sensor_flag=0;          //置限位标志位为 0
            }
        }
    }
}

void manu_input()                       //手动输入函数
{
    if(K5==0&&sensor_flag!=1)       //正常运行/右限位状态下按下左移按键
    {
        motor_flag=1;                   //置电机标志位为 1
        sensor_flag=0;                  //置限位标志位为 0
    }
    else if(K8==0&&sensor_flag!=2)  //正常运行/左限位状态下按下右移按键
    {
```

```
            motor_flag=2;                   //置电机标志位为 2
            sensor_flag=0;                  //置限位标志位为 0
        }
        else                                //左移/右移按键松开
        {
            motor_flag=0;                   //置电机标志位为 0
        }
}

void auto_input()                           //自动输入函数
{
    K5_last=K5_now;
    K5_now=K5;                              //启动按键键值刷新
    K8_last=K8_now;
    K8_now=K8;                              //停止按键键值刷新
    if(K5_last==0&&K5_now==1)                       //启动按键按下后松开
    {
        motor_flag=1;                       //置电机标志位为 1
    }
    else if(K8_last==0&&K8_now==1)                  //停止按键按下后松开
    {
        motor_flag=0;                       //置电机标志位为 0
    }
}

void output()                               //电机输出函数
{
    if(motor_flag==0)                       //电机标志位为 0
    {
        stop();                             //电机停止
    }
    else if(motor_flag==1)                  //电机标志位为 1
    {
        fwd();                              //电机正转
    }
    else if(motor_flag==2)                  //电机标志位为 2
    {
        back();                             //电机反转
    }
}

void display()                              //液晶显示函数
{
    if(motor_flag==0)                       //电机标志位为 0
```

```
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_stop);     //显示电机状态 stop
    }
    else if(motor_flag==1)              //电机标志位为 1
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_fwd);      //显示电机状态 fwd
    }
    else if(motor_flag==2)              //电机标志位为 2
    {
        lcd_pos(2,6);
        lcdwrite_string(disp_back);     //显示电机状态 back
    }
    if(mode_flag==0)                    //模式标志位为 0
    {
        lcd_pos(2,1);
        lcdwrite_string(disp_manu);     //显示系统模式 manu
        if(sensor_flag==0)              //限位标志位为 0
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_norm);//显示运行情况 norm
        }
        else if(sensor_flag==1)         //限位标志位为 1
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_senl); //显示运行情况 senl
        }
        else if(sensor_flag==2)         //限位标志位为 2
        {
            lcd_pos(2,11);
            lcdwrite_string(disp_senr); //显示运行情况 senr
        }
    }
    else                                //模式标志位为 1
    {
        lcd_pos(2,1);
        lcdwrite_string(disp_auto);     //显示系统模式 auto
        lcd_pos(2,11);
        lcdwrite_string(disp_no);
        lcdwrite_sz(count/10);
        lcdwrite_sz(count%10);          //显示运行次数 count
    }
}
```

```
void int_ex0() interrupt 0              //外部中断 0 函数
{
    if(mode_flag==0)                    //模式标志位为 0
    {
        stop();                         //电机停止
        if(motor_flag==1)               //电机标志位为 1
        {
            sensor_flag=1;              //置限位标志位为 1
        }
        else if(motor_flag==2)          //电机标志位为 2
        {
            sensor_flag=2;              //置限位标志位为 2
        }
        motor_flag=0;                   //置电机标志位为 0
    }
    else                                //模式标志位为 1
    {
        if(motor_flag==1)               //电机标志位为 1
        {
            motor_flag=2;               //置电机标志位为 2
        }
        else if(motor_flag==2)          //电机标志位为 2
        {
            motor_flag=1;               //置电机标志位为 1
            count++;                    //运行次数计数
        }
    }
}

void int_ex1() interrupt 2              //外部中断 1 函数
{
    stop();                             //电机停止
    motor_flag=0;                       //置电机标志位为 0
    limit_flag=1;                       //置急停标志位为 1
}
```

5.本章小结

本章设计开发了一种单轴运动工作台控制系统，主要涉及直流电机、独立按键、液晶显示的综合应用，并通过外部中断触发机制实现左右限位功能和紧急停止功能的快速响应和及时处理。使用外部中断存在一些注意事项和使用技巧，总

结如下：（1）当外部中断发出的中断请求被允许时，单片机将会响应该中断请求并自动跳转执行中断服务程序，因此中断触发按键不需要进行检测，中断服务程序也不需要进行声明和调用，进行中断初始化配置并使用关键字 interrupt 和中断号 n 建立中断服务程序即可；（2）为了快速处理中断请求以便继续响应其他请求，通常情况下中断服务程序根据功能要求仅改变标志位数值，回到系统主程序再根据标志位数值执行相应系统输出。同时，本章展示了综合控制系统的程序设计方法，灵活使用标志位划分系统状态，确定软件架构并搭建程序框架，然后依次进行各个模块的程序设计。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用外部中断的控制功能，并能够独立完成综合控制系统的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，增加单轴运动工作台的电机调速功能，要求能够通过相应按键设置电机速度，同时不能使用延时函数实现电机输出，具体功能自拟。

一种质量检测流水线控制系统开发

1.学习目标

1. 能根据控制系统任务要求完成直流电机、独立按键、液晶显示的综合应用；
2. 能应用定时器/计数器实现定时/计数控制和电机调速控制；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

在单片机应用中，常常会涉及定时或计数功能，例如对外部脉冲进行计数、产生精确的定时时间、实现周期性任务调度等。AT89C52 单片机共有 2 个定时器/计数器，可以通过寄存器 TMOD 选择定时模式或计数模式。此外，定时/计数溢出时会产生中断请求，如果中断请求被允许的话，单片机将会响应该中断请求并自动跳转执行中断服务程序。本章提出一例“设计一种质量检测流水线控制系统”的课题需求，学习定时器/计数器的初始设置和使用方法，并应用前面章节所学知识完成控制系统的设计工作。

任务题目：设计一种质量检测流水线控制系统

功能要求：该系统能够统计检测件数并调整电机速度，从而实现流水线产品输送和质量检测的协调控制。同时，在系统运行过程中能够按下暂停按键停止运行，按下启动按键重新启动。分别设定确定按键、暂停按键、启动按键，系统独立按键功能配置如表 5.1 所示。

表 5.1 系统独立按键功能配置表

	启动按键 K6		确定按键 K8
	暂停按键 K2		

具体控制要求如下：

1.初始界面显示。系统上电后在 1602LCD 第 1 行显示系统名称“Quality AL”（AL：Assembly Line 流水线），第 2 行显示欢迎标语“U ARE WELLCOME”。

按下确定按键，进入主控初始界面：第 1 行显示检测件数“check num:00/5s”，第 2 行显示电机速度“motor spd:00”。

2.流水线工作控制。根据该流水线的统计数据，每件产品进行质量检测一般需要 0.5~1s 的时间，要求如果质量检测的速度加快，则产品输送的速度也随之加快，反之亦然。其中流水线控制电机的占空比速度与流水线近 5s 内的检测件数成线性关系，即电机占空比速度=5*近 5s 内检测件数。

系统以每秒一次的频率统计近 5s 内检测件数并根据公式调整电机占空比速度，同时液晶显示跟随系统运行同频刷新检测件数和电机速度。其中定时功能使用定时器 T0 实现，计数功能使用定时器 T1 实现。

3.流水线暂停控制。系统在工作状态，如果遇到突发情况，可以按下暂停按键（外部中断 1 按键触发），系统立即停止，显示界面不再刷新。当排除突发情况，可以按下启动按键，系统仍然按照暂停前的电机速度工作。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输入模块

单片机输入模块包括独立按键控制模块和外部信号发生模块。

（1）独立按键控制模块主要用于系统运行过程中按下确定按键进入系统和按下启动按键重新启动各控制功能的人机操作。单片机通过判断 I/O 接口的电平状态，能够识别对应按键是否按下或松开，其控制原理已在第二章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P1.5、P1.7 引脚。

（2）外部信号发生模块主要用于提供外部中断所需的触发信号和定时器所需的计数信号。该模块的选型应根据实际控制装置的要求进行，本例不作展开。其中外部中断使用 P3.2、P3.3 引脚，在开发板上接有按键 K1、K2，当按键按下时会产生下降沿信号从而触发外部中断，其工作原理已在第四章案例中详细讲解，

此处不作赘述。

定时器/计数器有定时模式和计数模式两种工作模式，本质上都是对脉冲信号进行计数，定时模式是对内部时钟信号进行计数，计数模式是对外部输入信号进行计数。当定时器/计数器处于计数模式时，接口电路如图 5.1 所示，其中定时器 T0 使用 P3.4 引脚，在开发板上接有按键 K3，定时器 T1 使用 P3.5 引脚，在开发板上接有按键 K4。以定时器 T0 为例，当 K3 按下时 P3.4 会产生下降沿信号，单片机自动进行采样，定时器 T0 对该信号计数增 1。

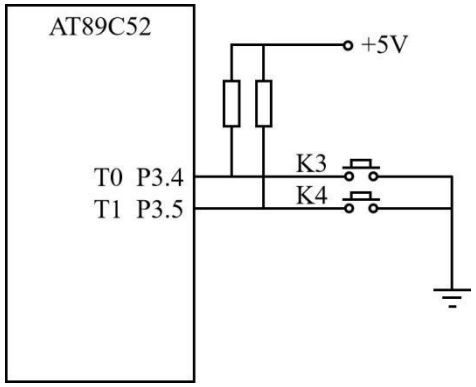


图 5.1 定时器/计数器接口电路图

3.2 单片机输出模块

单片机输出模块包括直流电机控制模块和液晶显示控制模块。

（1）直流电机控制模块主要用于流水线产品输送的动力驱动。该功能由单片机输出信号控制电机驱动，对旋转角度的控制要求不高，可以选择直流电机及其驱动模块的控制方案。同时该模块的选型应根据实际控制装置的要求进行，本例不作展开。直流电机的控制原理已在第一章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P2.6/P2.7 引脚。

（2）液晶显示控制模块主要用于系统运行过程中各项信息数据的实时刷新显示。单片机通过输出数据和命令至 1602LCD，从而实现相应的数字、字母、符号显示，其控制原理已在第三章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P0+P2.0/P2.1/P2.2 引脚。

3.3 单片机内部配置

单片机内部配置包括外部中断配置和定时器/计数器配置。

（1）外部中断配置主要用于外部信号发生模块产生暂停信号后的中断响应和中断处理，使用外部中断 1（P3.3）实现流水线暂停相关的控制功能。

（2）定时器/计数器配置主要用于系统运行过程中的时间控制和脉冲计数，其中定时器 T0 选择定时模式，实现定时相关的控制功能，定时器 T1（P3.5）选择计数模式，实现计数相关的控制功能。

系统端口及硬件配置如表 5.2 所示。

表 5.2 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P1.5	启动按键
2	P1.7	确定按键
3	P2.6/P2.7	直流电机
4	P0+P2.0/P2.1/P2.2	液晶显示
5	外部中断 1（P3.3）	暂停按键
6	定时器 T0	定时功能
7	定时器 T1（P3.5）	计数功能

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“系统等待”/“系统运行”和“流水线工作”/“流水线暂停”两组不同级别的模块，分别设定 bit 类型的系统标志位 system_flag 和暂停标志位 pause_flag，并将流水线模块置于系统模块的运

行状态下。系统上电后默认 system_flag=0 进入“系统等待”模块，通过按下确定按键置 system_flag=1 进入“系统运行”模块。当 pause_flag=0 时进入“流水线工作”模块，此时可执行统计检测件数和调整电机速度操作，当 pause_flag=1 时进入“流水线暂停”模块，通过按下启动按键置 pause_flag=0 回到“流水线工作”模块。

系统标志位对应关系如表 5.3 所示。

表 5.3 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
system_flag	0	系统等待
	1	系统运行
pause_flag	0	流水线工作
	1	流水线暂停

系统软件架构如图 5.2 所示。

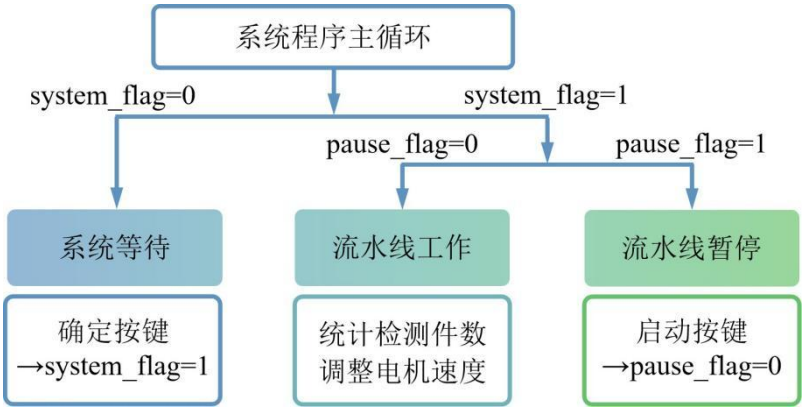


图 5.2 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由三个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 5.4 所示。

表 5.4 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	系统程序框架设计	标志位划分系统状态的应用
Step 2	流水线工作模块设计	定时器/计数器的应用
Step 3	流水线暂停模块设计	外部中断的应用

4.2.1 step1：系统程序框架设计

在进行控制系统各个模块的程序设计之前，搭建系统程序框架，即将软件架构以程序语言的形式体现。此外，系统上电后需要进行初始界面显示，放置在主函数的 while(1)循环前执行。

针对本步骤的设计任务，根据软件架构使用标志位划分系统状态以执行不同模块程序即可，同时根据功能要求调用液晶显示函数实现初始界面显示。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，其中“motor.h”为直流电机头文件，“1602LCD.h”为液晶显示头文件。定义 K8 为确定按键端口 P1.7，并初始化确定按键键值。定义 system_flag 为系统标志位，pause_flag 为暂停标志位：当系统标志位为 0 时，此时系统等待，按下确定按键置系统标志位为 1；当系统标志位为 1 时，此时系统运行，根据暂停标志位为 0 或 1 分别执行“流水线工作”或“流水线暂停”模块程序。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#define uchar unsigned char
sbit K8=P1^7;                    //确定按键端口 P1.7 定义
bit K8_last=1,K8_now=1;          //确定按键键值初始化
bit system_flag;                 //系统标志位定义
bit pause_flag;                  //暂停标志位定义

void main()                      //主函数
{
    while(1)
    {
        if(system_flag==0)       //系统标志位为 0
        {
```

```
            K8_last=K8_now;
            K8_now =K8;                 //确定按键键值刷新
            if(K8_last==0&&K8_now==1)           //确定按键按下后松开
            {
                system_flag=1;          //置系统标志位为 1
            }
        }
        else                            //系统标志位为 1
        {
            if(pause_flag==0)           //暂停标志位为 0
            {
            }
            else                        //暂停标志位为 1
            {
            }
        }
    }
}
```

（2）定义初始界面显示所需的系统名称、欢迎标语、检测件数、电机速度为 code 型字符串数组：首先进行液晶显示初始化，指定第 1 行第 1 列显示系统名称 name，第 2 行第 1 列显示欢迎标语 disp，然后按下确定按键进行状态切换和界面刷新，指定第 1 行第 1 列显示检测件数 check，第 2 行第 1 列显示电机速度 motor。相应程序如下：

```
...
uchar code name[] ={"     Quality AL     "};        //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};   //欢迎标语 disp 定义
uchar code check[]={"check num:00/5s "};          //检测件数 check 定义
uchar code motor[]={"motor spd:00       "};       //电机速度 motor 定义

void main()                             //主函数
{
    lcd_init();                         //液晶显示初始化
    lcd_pos(1,1);                       //显示位置第 1 行第 1 列
    lcdwrite_string(name);              //显示系统名称 name
    lcd_pos(2,1);                       //显示位置第 2 行第 1 列
    lcdwrite_string(disp);              //显示欢迎标语 disp
    while(1)
    {
        if(system_flag==0)              //系统标志位为 0
        {
            K8_last=K8_now;
            K8_now =K8;                 //确定按键键值刷新
```

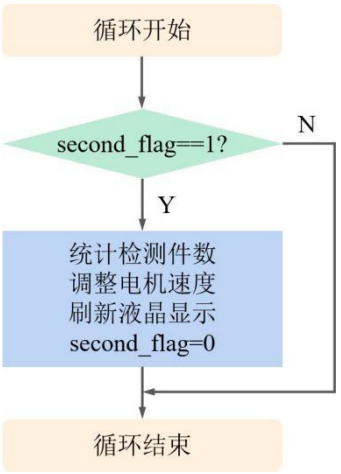
```
                if(K8_last==0&&K8_now==1)          //确定按键按下后松开
                {
                    system_flag=1;          //置系统标志位为 1
                    lcd_pos(1,1);
                    lcdwrite_string(check);//显示检测件数 check
                    lcd_pos(2,1);
                    lcdwrite_string(motor);//显示电机速度 motor
                }
            }
            else                            //系统标志位为 1
            {
            }
        }
    }
```

4.2.2 step2：流水线工作模块设计

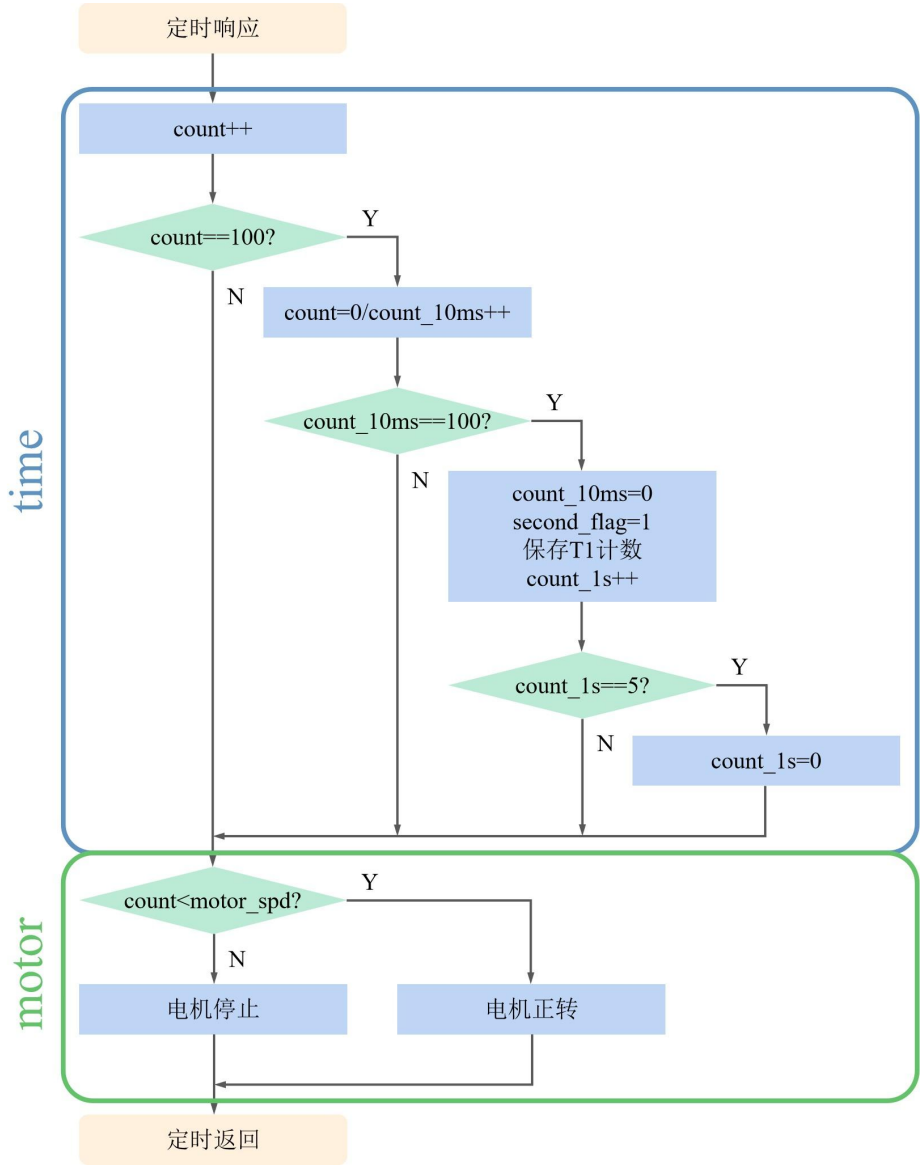
本步骤主要完成的具体任务是以每秒一次的频率统计近 5s 内检测件数并调整电机占空比速度，同时液晶显示同频刷新检测件数和电机速度。其中定时功能使用定时器 T0 实现，计数功能使用定时器 T1 实现。

分析具体任务中与定时或计数相关的控制功能。与定时相关的控制功能包括定时 1s 和定时 5s，定时 1s 后执行统计检测件数、调整电机速度、刷新液晶显示操作，定时 5s 后结束时间周期重新开始定时。此外，直流电机通过设定工作周期并按照占空比旋转和停止相应的时长来进行调速控制，其中涉及的时间控制同样使用定时器 T0 实现。与计数相关的控制功能包括检测件数计数，没有计数限制，不需要进入中断程序。

当定时器同时定时多个时间时，可以通过参数累加的方式进行。设定直流电机的工作周期为 10ms，按照 100%占空比划分定时时间为 100μs，则定时器每 100μs 产生溢出进入定时中断，累加至 100 为定时 10ms，累加至 100*100 为定时 1s，累加至 100*100*5 为定时 5s。此外，中断程序根据功能要求仅改变标志位数值，回到主程序再根据标志位数值执行相应系统输出。基于这点考虑，定时中断仅需要执行定时相关的电机输出，其余操作通过累加定时时间置相应标志位完成。设定 bit 类型的秒数标志位 second_flag，当定时中断定时 1s 时置 second_flag=1，此时主程序执行统计检测件数、调整电机速度、刷新液晶显示操作，执行结束置 second_flag=0。



(a)



(b)

图 5.3 流水线工作模块程序设计流程图

系统主程序的程序设计流程图如图 5.3(a)所示，定时器 T0 的程序设计流程图如图 5.3(b)所示，分为定时累加部分（time）和电机输出部分（motor），其中 count、count_10ms、count_1s 为定时累加参数。

针对本步骤的设计任务，设计思路应首先考虑实现定时器 T0 和定时器 T1 部分，进行定时/计数初始化配置，建立定时器 T0 的中断服务程序，并保存定时器 T1 的检测件数计数，然后在此基础上实现系统主程序部分，根据秒数标志位重新计算并刷新显示近 5s 内检测件数和电机占空比速度。

（1）使用定时器 T0 和定时器 T1 需要进行定时/计数初始化配置，包括设置寄存器 TMOD、设置定时/计数初值、打开中断允许开关，放置在主函数的 while(1)循环前执行。此外，还需要根据功能要求启动或停止定时器，如果定时器持续工作，则同样在初始化中启动即可。

寄存器 TMOD 用于选择定时器 T0 和定时器 T1 的工作模式和工作方式，其格式如图 5.4 所示。其中低 4 位（D0~D3）控制定时器 T0，高 4 位（D4~D7）控制定时器 T1。

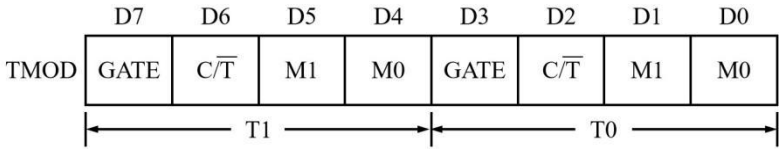


图 5.4 寄存器 TMOD 格式

GATE 为门控位。当 GATE=0 时，启动或停止定时器 T0、T1 由对应的控制位 TR0、TR1 控制；当 GATE=1 时，启动或停止定时器 T0、T1 由外部中断引脚电平和控制位 TR0、TR1 共同控制。通常设置门控位为 0，当 TR0=1、TR1=1 时启动定时器 T0、T1，当 TR0=0、TR1=0 时停止定时器 T0、T1。

C/$\overline{T}$ 为工作模式选择位。当 C/$\overline{T}$=0 时为定时模式，对内部时钟信号进行计数，其单个时间周期为 12 倍时钟周期，约为 1μs；当 C/$\overline{T}$=1 时为计数模式，对外部脉冲信号进行计数，其外部输入引脚分别为 P3.4、P3.5 引脚。

M1、M0 为工作方式选择位。根据 M1、M0 的不同设置，共有 4 种工作方式如表 5.5 所示。其中方式 0 为 13 位定时器/计数器，此时从初值计数至 2^{13} 产生溢出，需要使用指令重装初值；方式 1 为 16 位定时器/计数器，此时从初值计数至 2^{16} 产生溢出，需要使用指令重装初值；方式 2 为 8 位定时器/计数器，此时从初值计数至 2^{8} 产生溢出，溢出时自动重装初值；方式 3 只适用于定时器 T0，分

为 2 个 8 位定时器/计数器，一个使用寄存器 TMOD 的低 4 位控制，一个固定为 8 位定时使用 TR1 控制。

表 5.5 M1、M0 工作方式

M1	M0	工作方式	计数范围
0	0	方式 0，13 位定时器/计数器	$0\sim2^{13}$
0	1	方式 1，16 位定时器/计数器	$0\sim2^{16}$
1	0	方式 2，8 位自动重装定时器/计数器	$0\sim2^{8}$
1	1	方式 3，T0 分为 2 个 8 位定时器/计数器	$0\sim2^{8}$

已知定时器 T0 选择定时模式，定时时间为 100μs，定时器 T1 选择计数模式，没有计数限制。因此，可以设置定时器 T0 为方式 2 定时，定时器 T1 为方式 1 计数，此时寄存器 TMOD 为 01010010，转换为十六进制为 0x52。此外，设置定时器 T0 的定时初值 TL0、TH0 为 0xA4，定时器 T1 的计数初值 TL1、TH1 为 0。所涉及的定时初值可以在 STC 软件的定时器计算器中得到，设置系统频率为 11.0592MHz，选择定时器时钟为 12T（FOSC/12），如图 5.5 所示。

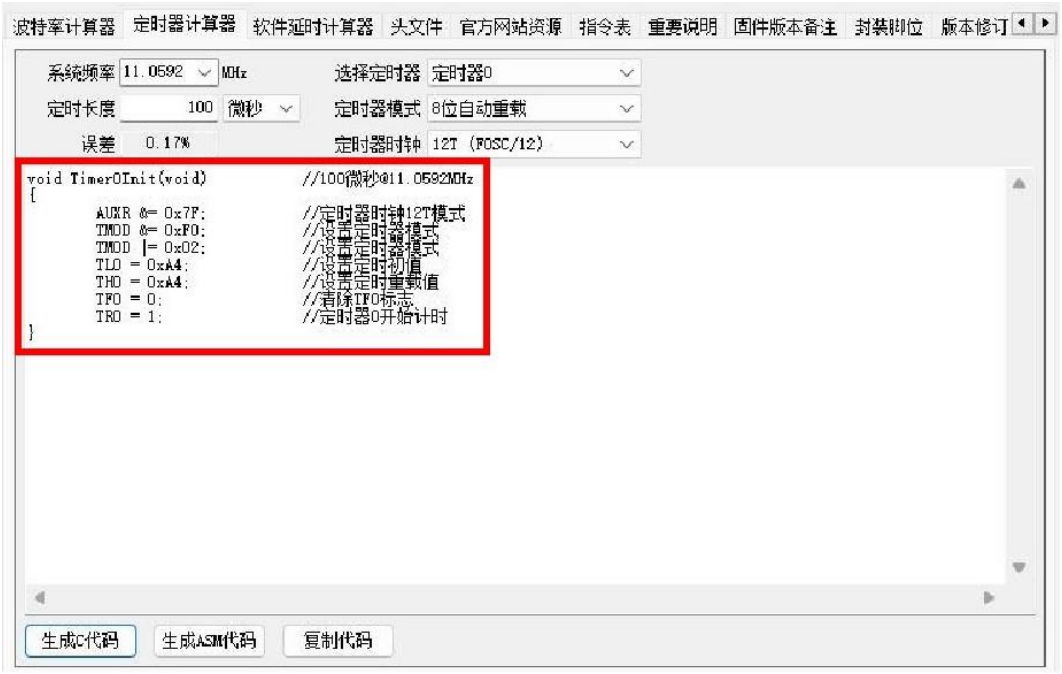


图 5.5 STC 定时器计算器界面

设置寄存器 TMOD 和定时/计数初值之后，定时/计数溢出时产生中断请求触发中断响应进行中断处理同样需要打开中断允许开关和建立中断服务程序。其中定时器 T0 开关为 ET0=1，定时器 T1 开关为 ET1=1。同时定时器 T0 的中断号为

1，定时器 T1 的中断号为 3，以定时器 T0 为例，可以建立中断服务程序为 void timer0() interrupt 1。

定义 second_flag 为秒数标志位，定义 count、count_10ms、count_1s 为定时累加参数，定义 number[5]为计数保存数组，定义 motor_spd 为电机速度。依次进行定时/计数初始化配置，按下确定按键启动定时器 T0、T1，并根据流水线工作模块程序设计流程图建立定时器 T0 函数。相应程序如下：

```
...
bit second_flag;                        //秒数标志位定义
uchar count,count_10ms,count_1s;        //定时累加参数定义
uchar number[5];                        //计数保存数组定义
uchar motor_spd;                        //电机速度定义

void main()
{
    TMOD=0x52;                          //设置定时器 T0 为方式 2 定时
                                          设置定时器 T1 为方式 1 计数
    TL0=0xA4;
    TH0=0xA4;                           //设置定时器 T0 初值
    TL1=0;
    TH1=0;                              //设置定时器 T1 初值
    EA=1;                               //打开总中断允许开关
    ET0=1;                              //打开定时器 T0 开关
    while(1)
    {
        if(system_flag==0)              //系统标志位为 0
        {
            K8_last=K8_now;
            K8_now =K8;                 //确定按键键值刷新
            if(K8_last==0&&K8_now==1)           //确定按键按下后松开
            {
                ...
                TR0=1;                  //启动定时器 T0
                TR1=1;                  //启动定时器 T1
            }
        }
        else                            //系统标志位为 1
        {
        }
    }
}
```

```
void timer0() interrupt 1                //定时器 T0 函数
{
    count++;                             //定时累加
    if(count==100)                       //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==100)              //定时 1s
        {
            count_10ms=0;
            second_flag=1;               //置秒数标志位为 1
            number[count_1s]=TL1;        //保存定时器 T1 计数
            TL1=0;                       //定时器 T1 计数清零
            count_1s++;
            if(count_1s==5)              //定时 5s
            {
                count_1s=0;
            }
        }
    }
    if(count<motor_spd)                  //电机输出
    {
        fwd();                           //电机正转
    }
    else
    {
        stop();                          //电机停止
    }
}
```

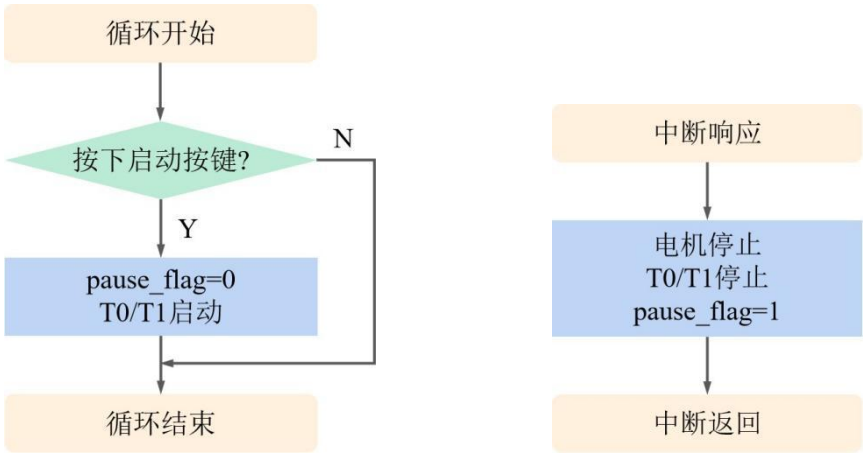
（2）在此基础上继续完成系统主程序部分，定义 check_num 为检测件数。当秒数标志位为 1 时，首先通过使用 if 语句循环累加保存数组的各位数值得到近 5s 内检测件数，然后根据对应关系计算电机占空比速度并限制其速度上限为 99，此时仅需要刷新显示界面中检测件数和电机速度的数值部分，最后置秒数标志位为 0。相应程序如下：

```
...
uchar check_num;                         //检测件数定义
void calcu_num();                        //计算检测件数函数声明

void main()                              //主函数
{
    ...
    while(1)
```

```
    {
        if(system_flag==0)              //系统标志位为 0
        {
        }
        else                            //系统标志位为 1
        {
            if(pause_flag==0)           //暂停标志位为 0
            {
                if(second_flag==1)      //秒数标志位为 1
                {
                    calcu_num();        //计算检测件数
                    motor_spd=5*check_num;      //计算电机速度
                    if(motor_spd>=100)          //限制电机速度
                    {
                        motor_spd=99;
                    }
                    lcd_pos(1,11);      //显示位置第 1 行第 11 列
                    lcdwrite_sz(check_num/10);
                    lcdwrite_sz(check_num%10);  //显示检测件数
                    lcd_pos(2,11);      //显示位置第 2 行第 11 列
                    lcdwrite_sz(motor_spd/10);
                    lcdwrite_sz(motor_spd%10);  //显示电机速度
                    second_flag=0;      //置秒数标志位为 0
                }
            }
            else                        //暂停标志位为 1
            {
            }
        }
    }
}

void calcu_num()                        //计算检测件数函数
{
    uchar k1;
    check_num=0;
    for(k1=0;k1<5;k1++)
    {
        check_num+=number[k1];
    }
}
```

4.2.3 step3：流水线暂停模块设计

本步骤主要完成的具体任务是在系统运行过程中能够按下暂停按键停止运行，按下启动按键重新启动。通过控制位 TR0、TR1 启动或停止定时器 T0、T1，其中系统主程序需要实现启动按键相关的控制功能，程序设计流程图如图 5.6(a)所示，外部中断 1 需要实现暂停按键相关的控制功能，程序设计流程图如图 5.6(b)所示。



(a) (b)

图 5.6 流水线暂停模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现外部中断 1 部分，进行中断初始化配置，建立中断服务程序，置暂停标志位并停止定时器，然后在此基础上实现系统主程序部分，检测启动按键是否按下，置暂停标志位并启动定时器。

（1）进行中断初始化配置，打开外部中断 1 开关并选择中断触发方式，根据流水线暂停模块程序设计流程图建立外部中断 1 函数。相应程序如下：

```
void main()                          //主函数
{
    EA=1;                            //打开总中断允许开关
    EX1=1;                           //打开外部中断 1 开关
    IT1=1;                           //选择跳沿触发方式
    while(1)
    {
    }
}

void int_ex1() interrupt 2           //外部中断 1 函数
{
```

```
    stop();                          //电机停止
    TR0=0;                           //停止定时器 T0
    TR1=0;                           //停止定时器 T1
    pause_flag=1;                    //置暂停标志位为 1
}
```

（2）在此基础上继续完成系统主程序部分，定义 K6 为启动按键端口 P1.5，并初始化解除按键键值。当启动按键按下后松开，置暂停标志位为 0 回到流水线工作状态，同时启动定时器 T0、T1 开始定时和计数。相应程序如下：

```
...
sbit K6=P1^5;                        //启动按键端口 P1.5 定义
bit K6_last=1,K6_now=1;              //启动按键键值初始化

void main()                          //主函数
{
    ...
    while(1)
    {
        if(system_flag==0)           //系统标志位为 0
        {
        }
        else                         //系统标志位为 1
        {
            if(pause_flag==0)        //暂停标志位为 0
            {
            }
            else                     //暂停标志位为 1
            {
                K6_last=K6_now;
                K6_now =K6;          //启动按键键值刷新
                if(K6_last==0&&K6_now==1)      //启动按键按下后松开
                {
                    pause_flag=0;    //置暂停标志位为 0
                    TR0=1;           //启动定时器 T0
                    TR1=1;           //启动定时器 T1
                }
            }
        }
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#define uchar unsigned char
uchar code name[] ={"      Quality AL      "};          //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};    //欢迎标语 disp 定义
uchar code check[]={"check num:00/5s "};          //检测件数 check 定义
uchar code motor[]={"motor spd:00       "};          //电机速度 motor 定义
sbit K6=P1^5;                          //启动按键端口 P1.5 定义
bit K6_last=1,K6_now=1;                //启动按键键值初始化
sbit K8=P1^7;                          //确定按键端口 P1.7 定义
bit K8_last=1,K8_now=1;                //确定按键键值初始化
bit system_flag;                       //系统标志位定义
bit pause_flag;                        //暂停标志位定义
bit second_flag;                       //秒数标志位定义
uchar count,count_10ms,count_1s;       //定时累加参数定义
uchar number[5];                       //计数保存数组定义
uchar check_num;                       //检测件数定义
uchar motor_spd;                       //电机速度定义
void calcu_num();                      //计算检测件数函数声明

void main()                            //主函数
{
    lcd_init();                        //液晶显示初始化
    lcd_pos(1,1);                      //显示位置第 1 行第 1 列
    lcdwrite_string(name);             //显示系统名称 name
    lcd_pos(2,1);                      //显示位置第 2 行第 1 列
    lcdwrite_string(disp);             //显示欢迎标语 disp
    TMOD=0x52;                         //设置定时器 T0 为方式 2 定时
                                         设置定时器 T1 为方式 1 计数
    TL0=0xA4;
    TH0=0xA4;                          //设置定时器 T0 初值
    TL1=0;
    TH1=0;                             //设置定时器 T1 初值
    EA=1;                              //打开总中断允许开关
    ET0=1;                             //打开定时器 T0 开关
    EX1=1;                             //打开外部中断 1 开关
    IT1=1;                             //选择跳沿触发方式
    while(1)
```

```
{
    if(system_flag==0)              //系统标志位为 0
    {
        K8_last=K8_now;
        K8_now =K8;                 //确定按键键值刷新
        if(K8_last==0&&K8_now==1)           //确定按键按下后松开
        {
            system_flag=1;          //置系统标志位为 1
            lcd_pos(1,1);
            lcdwrite_string(check);//显示检测件数 check
            lcd_pos(2,1);
            lcdwrite_string(motor);//显示电机速度 motor
            TR0=1;                  //启动定时器 T0
            TR1=1;                  //启动定时器 T1
        }
    }
    else                            //系统标志位为 1
    {
        if(pause_flag==0)           //暂停标志位为 0
        {
            if(second_flag==1)      //秒数标志位为 1
            {
                calcu_num();        //计算检测件数
                motor_spd=5*check_num;      //计算电机速度
                if(motor_spd>=100)          //限制电机速度
                {
                    motor_spd=99;
                }
                lcd_pos(1,11);      //显示位置第 1 行第 11 列
                lcdwrite_sz(check_num/10);
                lcdwrite_sz(check_num%10);    //显示检测件数
                lcd_pos(2,11);      //显示位置第 2 行第 11 列
                lcdwrite_sz(motor_spd/10);
                lcdwrite_sz(motor_spd%10);    //显示电机速度
                second_flag=0;      //置秒数标志位为 0
            }
        }
        else                        //暂停标志位为 1
        {
            K6_last=K6_now;
            K6_now =K6;             //启动按键键值刷新
            if(K6_last==0&&K6_now==1)       //启动按键按下后松开
            {
                pause_flag=0;       //置暂停标志位为 0
```

```
                    TR0=1;              //启动定时器 T0
                    TR1=1;              //启动定时器 T1
                }
            }
        }
    }
}

void calcu_num()                        //计算检测件数函数
{
    uchar k1;
    check_num=0;
    for(k1=0;k1<5;k1++)
    {
        check_num+=number[k1];
    }
}

void timer0() interrupt 1               //定时器 T0 函数
{
    count++;                            //定时累加
    if(count==100)                      //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==100)             //定时 1s
        {
            count_10ms=0;
            second_flag=1;              //置秒数标志位为 1
            number[count_1s]=TL1;       //保存定时器 T1 计数
            TL1=0;                      //定时器 T1 计数清零
            count_1s++;
            if(count_1s==5)             //定时 5s
            {
                count_1s=0;
            }
        }
    }
    if(count<motor_spd)                 //电机输出
    {
        fwd();                          //电机正转
    }
    else
    {
```

```
            stop();                         //电机停止
        }
    }

    void int_ex1() interrupt 2              //外部中断 1 函数
    {
        stop();                             //电机停止
        TR0=0;                              //停止定时器 T0
        TR1=0;                              //停止定时器 T1
        pause_flag=1;                       //置暂停标志位为 1
    }
```

5.本章小结

本章设计开发了一种质量检测流水线控制系统，结合前面章节所学知识使用定时器/计数器及其中断服务程序实现流水线产品的检测计数和电机的自动调速。使用定时器/计数器存在一些注意事项和使用技巧，总结如下：（1）定时/计数初始化配置包括设置寄存器 TMOD、设置定时/计数初值、打开中断允许开关、启动定时器/计数器，根据功能要求灵活进行配置；（2）通常情况下，可以将定时中断分为定时和输出两部分，其中定时部分通过参数累加的方式进行多个时间的定时，对应定时时间改变标志位数值，输出部分执行定时相关的系统输出；（3）当定时器持续工作时，定时开始需要清零一系列累加参数以得到精确的定时时间，这在后面章节的案例中将有所体现。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用定时器/计数器的控制功能，在主函数的 while(1) 循环中不再使用延时函数。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，增加质量检测流水线的断电保持功能，要求系统断电时保存检测件数和电机速度，恢复供电后按照原有数据和状态工作，具体功能自拟。

一种太空电梯控制系统开发

1.学习目标

1. 能根据控制系统任务要求完成矩阵按键的键值设置和扫描判断；
2. 能应用存储器 EEPROM 实现控制系统数据的断电保持功能；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

随着控制系统越来越复杂，所需按键数目越来越多，使用独立按键控制会占用较多的 I/O 接口。矩阵按键由行列结构交叉构成，其中一个 8 位的并行 I/O 接口即可构成一个 4*4 结构的矩阵式键盘。因此，在所需按键数目较多的情况下，使用矩阵按键控制以节省单片机的端口资源，其头文件见附录“matrix_key.h”。此外，可以使用存储器 EEPROM 读取或写入数据从而实现断电保持功能，其头文件见附录“stc89c52_eeprom.h”。

本章提出一例“设计一种太空电梯控制系统”的课题需求，学习矩阵按键的键值设置和扫描判断，学习 EEPROM 的调用函数和使用方法，并应用前面章节所学知识完成控制系统的设计工作。

任务题目：设计一种太空电梯控制系统

功能要求：该系统能够设置电机转向、电机速度、运行时间，并控制太空电梯按照设置参数运行，从而实现地球和太空之间的连接传送。系统在电机停止的情况下可以按下功能按键切换参数设置功能和电梯运行功能，其中系统独立按键功能配置如表 6.1 所示，系统矩阵按键功能配置如表 6.2 所示。

表 6.1 系统独立按键功能配置表

人员进入 K1	人员离开 K2		功能按键 K4

表 6.2 系统矩阵按键功能配置表

转向设置	1	2	3
速度设置	4	5	6
时间设置	7	8	9
启动	保存	0	确定

具体控制要求如下：

1.初始界面显示。系统上电后在 1602LCD 第 1 行显示系统名称“TIANTI”，第 2 行显示欢迎标语“U ARE WELLCOME”。1 秒后清屏，进入主控初始界面：第 1 行依次显示电机转向、电机速度、运行时间信息“*fwd S:50 T:3”，第 2 行依次显示运行状态、搭乘人数信息“stop number:00”，其中符号*用于区分当前功能。

2.参数设置功能。

转向设置：系统上电后默认电机转向为正转，按下转向设置按键，切换电机转向为正转或反转，同时显示界面刷新显示。

速度设置：系统上电后默认电机速度为 50%，按下速度设置按键，显示界面对应电机速度位置显示“S(__)”，此时按下 0~9 数字键输入电机速度，其参数设置范围为 1~99。按下确定按键检查参数，如果在设置范围内则覆盖数值并刷新显示。

时间设置：系统上电后默认运行时间为 3s，按下时间设置按键，显示界面对应运行时间位置显示“T(_)”，此时按下 0~9 数字键输入运行时间，其参数设置范围为 1~9。按下确定按键检查参数，如果在设置范围内则覆盖数值并刷新显示。

参数保存：按下保存按键，使用 EEPROM 保存电机速度和运行时间数据，系统断电开机后仍然使用保存的原有数据。

3.电梯运行功能。使用激光传感器检测人员进入和离开电梯（外部中断 0、外部中断 1 按键触发），同时记录人数并刷新显示。当搭乘人数为 1~10 人时，按下启动按键，电梯按照设置参数运行，当搭乘人数超过 10 人时，蜂鸣器报警 1s 提醒电梯超载，其中定时功能使用定时器 T0 实现。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输入模块

单片机输入模块包括独立按键控制模块、矩阵按键控制模块和外部信号发生模块。

（1）独立按键控制模块主要用于按下功能按键切换参数设置功能和电梯运行功能。单片机通过判断 I/O 接口的电平状态，能够识别对应按键是否按下或松开，其控制原理已在第二章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P3.5 引脚。

（2）矩阵按键控制模块主要用于系统运行过程中转向设置、速度设置、时间设置、参数保存、电梯启动各控制功能的人机操作。已知矩阵按键由行列结构交叉构成，当任意按键按下时行线和列线导通，单片机通过扫描行线信号和列线信号能够识别按键状态和按键位置：首先置列线为低电平，置行线为高电平，此时按键按下所处行线为低电平，然后置列线为高电平，置行线为低电平，此时按键按下所处列线为低电平。矩阵按键接口电路如图 6.1 所示，在开发板上占用 P1 引脚，其中 P1.0~P1.3 为行线，P1.4~P1.7 为列线，因此各个位置对应行列信号如表 6.3 所示。以第 1 行第 1 列按键为例，当按键按下时 P1=11101110=0xee，当按键松开时 P1=11111111=0xff。

表 6.3 矩阵按键行列信号

0xee	0xde	0xbe	0x7e
0xed	0xdd	0xbd	0x7d
0xeb	0xdb	0xbb	0x7b
0xe7	0xd7	0xb7	0x77
按键松开：0xff			

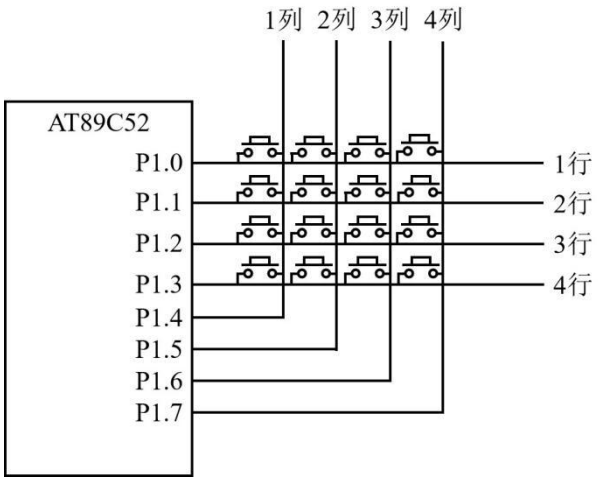


图 6.1 矩阵按键接口电路图

（3）外部信号发生模块主要用于提供人员进入和离开信号，产生中断请求触发外部中断 0 和外部中断 1。该模块的选型应根据实际控制装置的要求进行，本例不作展开。其中外部中断使用 P3.2、P3.3 引脚，在开发板上接有按键 K1、K2，当按键按下时会产生下降沿信号从而触发外部中断，其工作原理已在第四章案例中详细讲解，此处不作赘述。

3.2 单片机输出模块

单片机输出模块包括蜂鸣器控制模块、直流电机控制模块和液晶显示控制模块。

（1）蜂鸣器控制模块主要用于电梯超载时的报警提醒。该模块将脉冲信号输入转换为声音信号输出，通过调整脉冲信号的频率可以发出不同音调的声音。蜂鸣器接口电路如图 6.2 所示，在开发板上占用 P3.6 引脚。当 P3.6 以一定频率持续取反时蜂鸣器发声，当 P3.6=0 或 P3.6=1 时蜂鸣器不发声。

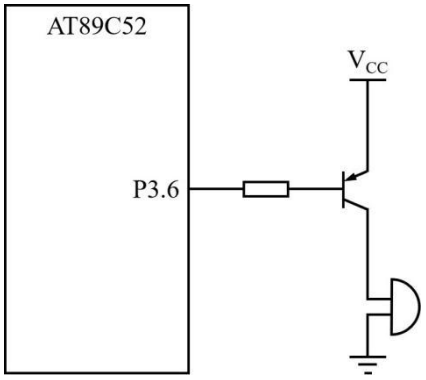


图 6.2 蜂鸣器接口电路图

（2）直流电机控制模块主要用于电梯运行时的动力驱动。该功能由单片机输出信号控制电机驱动，对旋转角度的控制要求不高，可以选择直流电机及其驱动模块的控制方案。同时该模块的选型应根据实际控制装置的要求进行，本例不作展开。直流电机的控制原理已在第一章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P2.6/P2.7 引脚。

（3）液晶显示控制模块主要用于系统运行过程中各项信息数据的实时刷新显示。单片机通过输出数据和命令至 1602LCD，从而实现相应的数字、字母、符号显示，其控制原理已在第三章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P0+P2.0/P2.1/P2.2 引脚。

3.3 单片机内部配置

单片机内部配置包括外部中断配置、定时器配置和 EEPROM 配置。

（1）外部中断配置主要用于外部信号发生模块产生人员进入和离开信号后的中断响应和中断处理，使用外部中断 0（P3.2）和外部中断 1（P3.3）实现搭乘人数相关的控制功能。

（2）定时器配置主要用于系统运行过程中的时间控制，其中定时器 T0 选择定时模式，实现定时相关的控制功能，包括电机调速控制和蜂鸣器发声控制。

（3）EEPROM 配置主要用于保存电机速度和运行时间数据，并在系统断电开机后提供保存的原有数据。

系统端口及硬件配置如表 6.4 所示。

表 6.4 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P1	矩阵按键
2	P3.5	功能按键
3	P3.6	蜂鸣器
4	P2.6/P2.7	直流电机
5	P0+P2.0/P2.1/P2.2	液晶显示
6	外部中断 0（P3.2）	人员进入
7	外部中断 1（P3.3）	人员离开

8	定时器 T0	定时功能
9	EEPROM	保存数据

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“参数设置”和“电梯运行”两个同一级别的模块，设定 bit 类型的状态标志位 status_flag，通过按下功能按键取反标志位，置 status_flag=~status_flag。当 status_flag=0 时进入“参数设置”模块，此时可执行转向设置、速度设置、时间设置、参数保存操作，当 status_flag=1 时进入“电梯运行”模块，通过按下启动按键控制电梯按照设置参数运行。

系统标志位对应关系如表 6.4 所示。

表 6.4 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
status_flag	0	参数设置
	1	电梯运行

系统软件架构如图 6.3 所示。

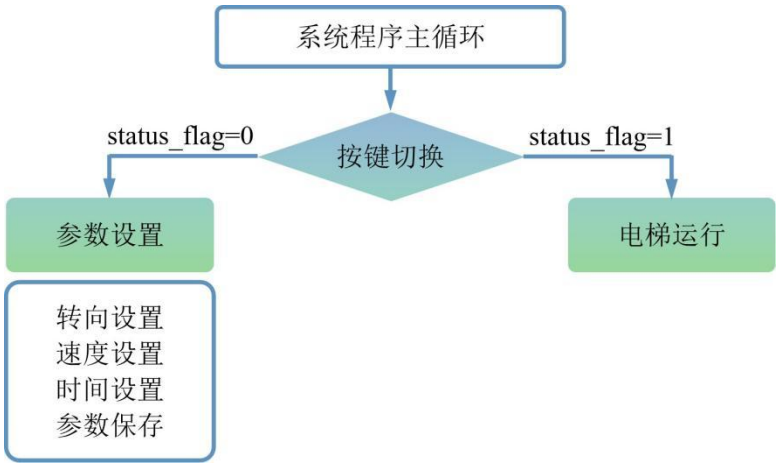


图 6.3 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由三个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 6.5 所示。

表 6.5 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	系统程序框架设计	标志位划分系统状态的应用
Step 2	参数设置模块设计	矩阵按键和 EEPROM 的应用
Step 3	电梯运行模块设计	外部中断和定时器的应用

4.2.1 step1：系统程序框架设计

在进行控制系统各个模块的程序设计之前，搭建系统程序框架，即将软件架构以程序语言的形式体现。此外，系统上电后需要进行初始界面显示，放置在主函数的 while(1)循环前执行。

针对本步骤的设计任务，根据软件架构使用标志位划分系统状态以执行不同模块程序即可，同时根据功能要求调用液晶显示函数实现初始界面显示。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，其中“motor.h”为直流电机头文件，“1602LCD.h”为液晶显示头文件，“matrix_key.h”为矩阵按键头文件，“stc89c52_eeprom.h”为 EEPROM 头文件。定义 K4 为功能按键端口 P3.5，并初始化确定按键键值。定义 status_flag 为状态标志位：按下功能按键取反状态标志位，根据状态标志位为 0 或 1 分别执行“参数设置”或“电梯运行”模块程序。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#include"matrix_key.h"
#include"stc89c52_eeprom.h"
#define uchar unsigned char
#define uint unsigned int
sbit K4=P3^5;                          //功能按键端口 P3.5 定义
```

```
bit K4_last=1,K4_now=1;                 //功能按键键值初始化
bit status_flag;                        //状态标志位定义

void main()                             //主函数
{
    while(1)
    {
        K4_last=K4_now;
        K4_now =K4;                     //功能按键键值刷新
        if(K4_last==0&&K4_now==1)               //功能按键按下后松开
        {
            status_flag=~status_flag;   //状态标志位取反
        }
        if(status_flag==0)              //状态标志位为 0
        {
        }
        else                            //状态标志位为 1
        {
        }
    }
}
```

（2）定义初始界面显示所需的系统名称、欢迎标语、电机转向、电机速度、运行时间、运行状态、搭乘人数为 code 型字符串数组，定义 speed 为电机速度，初值为 50，定义 time 为运行时间，初值为 3，定义 num 为搭乘人数，初值为 0，建立系统初始化函数：首先进行液晶显示初始化，指定第 1 行第 1 列显示系统名称 name，第 2 行第 1 列显示欢迎标语 disp，然后延时 1000ms 进行液晶显示清屏，指定第 1 行第 1 列依次显示符号*、电机转向 fwd、电机速度 speed、运行时间 time，第 2 行第 1 列依次显示空格、运行状态 stop、搭乘人数 num。此外，按下功能按键进行状态切换和界面刷新，当状态标志位为 0 时指定第 1 行第 1 列显示符号*，当状态标志位为 1 时指定第 2 行第 1 列显示符号*。相应程序如下：

```
...
uchar code name[] ={"        TIANTI        "};      //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};   //欢迎标语 disp 定义
uchar code disp_fwd[]    ={"fwd "};                 //电机转向 fwd 定义
uchar code disp_speed[]  ={" S:"};                  //电机速度 speed 定义
uchar code disp_time[]   ={" T:"};                  //运行时间 time 定义
uchar code disp_stop[]   ={"stop"};                 //运行状态 stop 定义
uchar code disp_num[]    ={" number:"};             //搭乘人数 num 定义
uchar speed=50;                         //电机速度定义
uchar time=3;                           //运行时间定义
```

```
uchar num=0;                        //搭乘人数定义
void init_sys();                    //系统初始化函数声明
void Delay1000ms();                 //延时 1000ms 函数声明

void main()                         //主函数
{
    init_sys();                     //系统初始化
    while(1)
    {
        K4_last=K4_now;
        K4_now =K4;                 //功能按键键值刷新
        if(K4_last==0&&K4_now==1)           //功能按键按下后松开
        {
            status_flag=~status_flag; //状态标志位取反
            if(status_flag==0)      //状态标志位为 0
            {
                lcd_pos(1,1);       //显示位置第 1 行第 1 列
                lcdwrite_zm('*');   //显示符号*
                lcd_pos(2,1);
                lcdwrite_zm(' ');
            }
            else                    //状态标志位为 1
            {
                lcd_pos(1,1);
                lcdwrite_zm(' ');
                lcd_pos(2,1);       //显示位置第 2 行第 1 列
                lcdwrite_zm('*');   //显示符号*
            }
        }
        if(status_flag==0)          //状态标志位为 0
        {
        }
        else                        //状态标志位为 1
        {
        }
    }
}

void init_sys()                     //系统初始化函数
{
    lcd_init();                     //液晶显示初始化
    lcd_pos(1,1);                   //显示位置第 1 行第 1 列
    lcdwrite_string(name);          //显示系统名称 name
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
```

```
    lcdwrite_string(disp);              //显示欢迎标语 disp
    Delay1000ms();                      //延时 1000ms
    lcd_clear();                        //液晶显示清屏
    lcd_pos(1,1);                       //显示位置第 1 行第 1 列
    lcdwrite_zm('*');                   //显示符号*
    lcdwrite_string(disp_fwd);          //显示电机转向 fwd
    lcdwrite_string(disp_speed);        //显示电机速度 speed
    lcdwrite_sz(speed/10);
    lcdwrite_sz(speed%10);              //显示电机速度
    lcdwrite_string(disp_time);         //显示运行时间 time
    lcdwrite_sz(time);                  //显示运行时间
    lcd_pos(2,1);                       //显示位置第 2 行第 1 列
    lcdwrite_zm(' ');                   //显示空格
    lcdwrite_string(disp_stop);         //显示运行状态 stop
    lcdwrite_string(disp_num);          //显示搭乘人数 num
    lcdwrite_sz(num/10);
    lcdwrite_sz(num%10);                //显示搭乘人数
}

void Delay1000ms()                      //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

4.2.2 step2：参数设置模块设计

本步骤涉及到同层级多任务控制，包括转向设置、速度设置、时间设置、参数保存，通过按下不同按键进行具体任务执行：按下转向设置按键，切换电机转向；按下速度设置按键，输入电机速度，按下确定按键检查并覆盖；按下时间设置按键，输入运行时间，按下确定按键检查并覆盖；按下保存按键，使用 EEPROM

保存数据。同时液晶显示跟随系统运行实时刷新显示，程序设计时需要注意显示位置、显示内容、显示覆盖等细节问题。

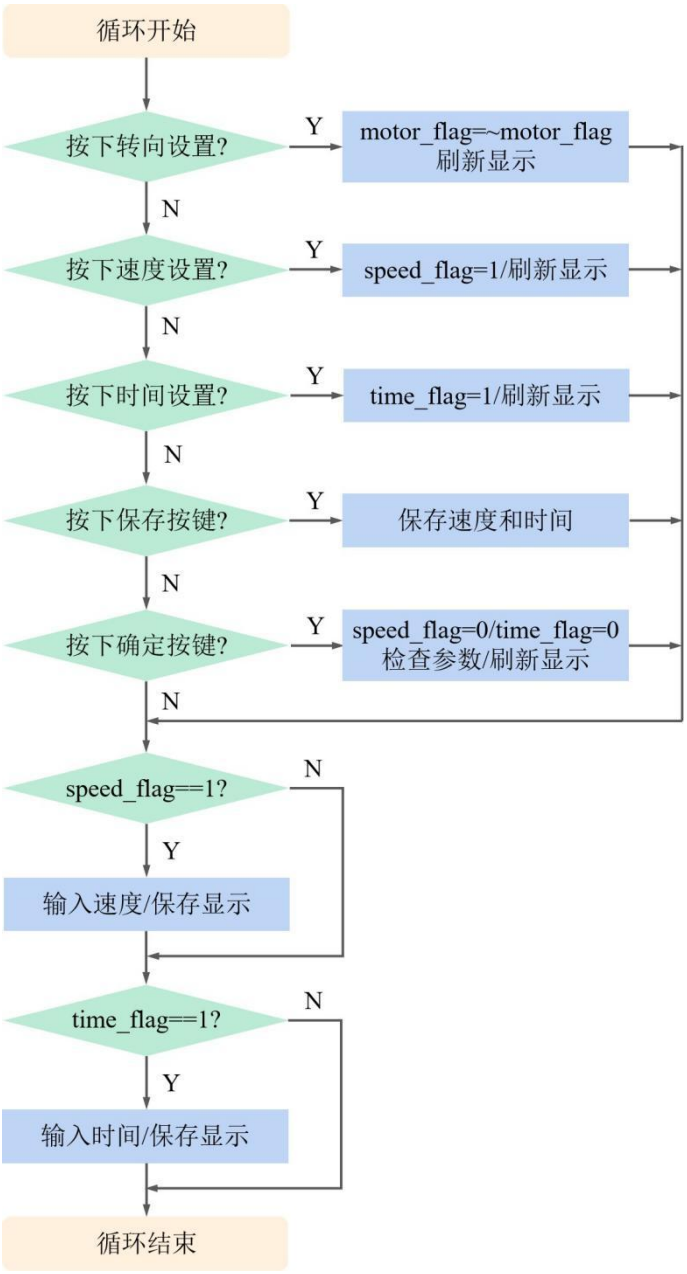


图 6.4 参数设置模块程序设计流程图

使用标志位梳理控制功能，界定系统状态。其中转向设置仅需要切换电机转向而不必控制电机输出，设定 bit 类型的转向标志位 motor_flag，通过按下转向设置按键取反标志位，置 motor_flag=~motor_flag，当 motor_flag=0 时为电机正转状态，当 motor_flag=1 时为电机反转状态。速度设置和时间设置需要实时检测数字键，放置在 while(1)循环中执行，分别设定 bit 类型的速度标志位 speed_flag 和时间标志位 time_flag，通过按下速度设置按键置 speed_flag=1 进入速度设置状

态，或是通过按下时间设置按键置 time_flag=1 进入时间设置状态，同时根据当前标志位判断按下确定按键置 speed_flag=0 退出速度设置状态或是置 time_flag=0 退出时间设置状态。参数保存能够跟随保存按键执行，调用 EEPROM 函数完成电机速度和运行时间数据的写入操作即可。系统标志位对应关系如表 6.6 所示，程序设计流程图如图 6.4 所示。

表 6.6 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
motor_flag	0	电机正转
	1	电机反转
speed_flag	0	退出速度设置
	1	进入速度设置
time_flag	0	退出时间设置
	1	进入时间设置

针对本步骤的设计任务，设计思路应考虑分任务完成，根据程序设计流程图依次实现转向设置、速度设置、时间设置、参数保存四个部分。此外，为了方便矩阵按键的使用，在“matrix_key.h”头文件中根据按键功能进行键值设置，此时调用按键检测函数扫描 P1 引脚的行列信号返回的是所设置的按键名称，其中 0~9 数字键可以命名为对应数字，各个功能键则可以命名为 0x10~0x60。以第 1 行第 1 列按键为例，该按键功能为转向设置，其键值设置为 0x10：当按键按下时 scan_key()=0x10，当按键松开时 scan_key()=0xff，当上一状态 key_last=0x10 且当前状态 key_now=0xff 时，表明按键按下后松开为一次按键有效。相应程序如下：

```
#include<reg51.h>
#define uchar unsigned char

uchar scan_key()
{
    uchar key_value,key_value1,key_value2,key_name;
    P1=0xf0;
    key_value1=P1;
    P1=0x0f;
    key_value2=P1;
    key_value=key_value1+key_value2;
```

```
    switch(key_value)
    {
        case 0xee: key_name=0x10;break;
        case 0xde: key_name=1;break;
        case 0xbe: key_name=2;break;
        case 0x7e: key_name=3;break;
        case 0xed: key_name=0x20;break;
        case 0xdd: key_name=4;break;
        case 0xbd: key_name=5;break;
        case 0x7d: key_name=6;break;
        case 0xeb: key_name=0x30;break;
        case 0xdb: key_name=7;break;
        case 0xbb: key_name=8;break;
        case 0x7b: key_name=9;break;
        case 0xe7: key_name=0x40;break;
        case 0xd7: key_name=0x50;break;
        case 0xb7: key_name=0;break;
        case 0x77: key_name=0x60;break;
        default:     key_name=0xff;
    }
    return(key_name);
}
```

（1）首先完成转向设置部分，定义所需的显示内容为 code 型字符串数组，定义矩阵按键并初始化其键值，定义 motor_flag 为转向标志位。在 while(1)循环中刷新矩阵按键键值，当转向设置按键按下后松开，取反转向标志位并刷新显示对应信息。相应程序如下：

```
...
uchar code disp_back[]   ={"back"};                  //电机转向 back 定义
uchar key_last=0xff,key_now=0xff;   //矩阵按键键值初始化
bit motor_flag;                     //转向标志位定义

void main()                         //主函数
{
    init_sys();                     //系统初始化
    while(1)
    {
        ...
        key_last=key_now;
        key_now=scan_key();         //矩阵按键键值刷新
        if(status_flag==0)          //状态标志位为 0
        {
            if(key_last==0x10&&key_now==0xff)  //转向设置按键按下后松开
```

```
            {
                motor_flag=~motor_flag;          //转向标志位取反
                if(motor_flag==0)                //转向标志位为 0
                {
                    lcd_pos(1,2);                //显示位置第 1 行第 2 列
                    lcdwrite_string(disp_fwd);   //显示电机转向 fwd
                }
                else                             //转向标志为为 1
                {
                    lcd_pos(1,2);                //显示位置第 1 行第 2 列
                    lcdwrite_string(disp_back);  //显示电机转向 back
                }
            }
        }
        else                                     //状态标志位为 1
        {
        }
    }
}
```

（2）然后完成速度设置部分，定义 speed_flag 为速度标志位，定义 set_speed[2] 为输入速度数组。当速度设置按键按下后松开，置速度标志位为 1 进入速度设置状态，实时检测 0~9 数字键并保存显示相应数字。当确定按键按下后松开且速度标志位为 1 时，置速度标志位为 0 退出速度设置状态，同时检查速度设置范围并刷新显示对应信息。相应程序如下：

```
...
bit speed_flag;                                  //速度标志位定义
uchar set_speed[2];                              //输入速度数组定义

void main()                                      //主函数
{
    init_sys();                                  //系统初始化
    while(1)
    {
        ...
        key_last=key_now;
        key_now=scan_key();                      //矩阵按键键值刷新
        if(status_flag==0)                       //状态标志位为 0
        {
            if(key_last==0x20&&key_now==0xff&&time_flag==0)
                                                 //速度设置按键按下后松开
            {
                speed_flag=1;                    //置速度标志位为 1
```

```
                lcd_pos(1,8);              //显示位置第 1 行第 8 列
                lcdwrite_zm('(');
                lcdwrite_zm('_');
                lcdwrite_zm('_');
                lcdwrite_zm(')');          //电机速度位置刷新显示
            }
            else if(key_last==0x60&&key_now==0xff)
                                           //确定按键按下后松开
            {
                if(speed_flag==1)          //速度标志位为 1
                {
                    speed_flag=0;          //置速度标志位为 0
                    if(set_speed[1]+set_speed[0]!=0)//检查速度设置范围
                    {
                        speed=set_speed[1]*10+set_speed[0];
                    }
                    lcd_pos(1,6);          //显示位置第 1 行第 6 列
                    lcdwrite_string(disp_speed);
                    lcdwrite_sz(speed/10);
                    lcdwrite_sz(speed%10);
                    lcdwrite_zm(' ');      //电机速度位置刷新显示
                }
            }
            if(speed_flag==1)              //速度标志位为 1
            {
                if(key_last<=9&&key_now==0xff)  //0~9 数字键按下后松开
                {
                    set_speed[1]=set_speed[0];
                    set_speed[0]=key_last;          //保存输入速度
                    lcd_pos(1,9);                   //显示位置第 1 行第 9 列
                    lcdwrite_sz(set_speed[1]);
                    lcdwrite_sz(set_speed[0]);      //显示输入速度
                }
            }
        }
        else                               //状态标志位为 1
        {
        }
    }
}
```

（3）继续完成时间设置部分，定义 time_flag 为时间标志位，定义 set_time 为输入时间。当时间设置按键按下后松开，置时间标志位为 1 进入时间设置状态，实时检测 0~9 数字键并保存显示相应数字。当确定按键按下后松开且时间标志位

为 1 时，置时间标志位为 0 退出时间设置状态，同时检查时间设置范围并刷新显示对应信息。相应程序如下：

```
...
bit time_flag;                          //时间标志位定义
uchar set_time;                         //输入时间定义

void main()                             //主函数
{
    init_sys();                         //系统初始化
    while(1)
    {
        ...
        key_last=key_now;
        key_now=scan_key();             //矩阵按键键值刷新
        if(status_flag==0)              //状态标志位为 0
        {
            if(key_last==0x30&&key_now==0xff&&speed_flag==0)
                                        //时间设置按键按下后松开
            {
                time_flag=1;            //置时间标志位为 1
                lcd_pos(1,13);          //显示位置第 1 行第 13 列
                lcdwrite_zm('(');
                lcdwrite_zm('_');
                lcdwrite_zm(')');       //运行时间位置刷新显示
            }
            else if(key_last==0x60&&key_now==0xff)
                                        //确定按键按下后松开
            {
                if(time_flag==1)        //时间标志位为 1
                {
                    time_flag=0;        //置时间标志位为 0
                    if(set_time!=0)     //检查时间设置范围
                    {
                        time=set_time;
                    }
                    lcd_pos(1,11);      //显示位置第 1 行第 11 列
                    lcdwrite_string(disp_time);
                    lcdwrite_sz(time);
                    lcdwrite_zm(' ');   //运行时间位置刷新显示
                }
            }
            if(time_flag==1)            //时间标志位为 1
            {
```

```
                if(key_last<=9&&key_now==0xff)  //0~9 数字键按下后松开
                {
                    set_time=key_last;          //保存输入时间
                    lcd_pos(1,14);              //显示位置第 1 行第 14 列
                    lcdwrite_sz(set_time);      //显示输入时间
                }
            }
        }
        else                                    //状态标志位为 1
        {
        }
    }
}
```

（4）最后完成参数保存部分。

EEPROM 调用函数如表 6.7 所示。其中 read_eeprom()为 EEPROM 读取函数，用于读取具体单元地址所存放的数据；erase_eeprom()为 EEPROM 擦除函数，在写入新的数据之前需要以扇区为单位擦除原有数据；store_eeprom()为 EEPROM 写入函数，用于将需要保存的指定数据写入具体单元地址。此外，EEPROM 为空时其各单元为 0xff。

表 6.7 EEPROM 调用函数

函数名称	函数说明
read_eeprom(add)	EEPROM 读取（读取地址 add 所存数据）
erase_eeprom(add)	EEPROM 擦除（擦除地址 add 所在扇区）
store_eeprom(add,1)	EEPROM 写入（将数值 1 写入地址 add）

使用 EEPROM 需要明确保存数据和单元地址，其中 AT89C52 的 EEPROM 共有八个扇区，第一扇区的起始地址为 0x2000，结束地址为 0x21ff，以此类推。在扇区中选择某一具体单元地址存放需要保存的指定数据，选择地址 0x2023 用于存放电机速度，后一地址 0x2024 用于存放运行时间。当保存按键按下后松开，调用擦除函数和写入函数完成两个数据的写入操作，同时设计开机后进入主函数即调用读取函数完成两个数据的读取操作，如果单元地址没有保存数据则使用初始数值，如果单元地址存有保存数据则更新参数数值，从而实现控制系统数据的断电保持功能。相应程序如下：

```
...
uint add=0x2023;                    //EEPROM 地址定义
```

```
void main()                          //主函数
{
    init_sys();                      //系统初始化
    while(1)
    {
        ...
        key_last=key_now;
        key_now=scan_key();          //矩阵按键键值刷新
        if(status_flag==0)           //状态标志位为 0
        {
            if(key_last==0x50&&key_now==0xff)
                                     //保存按键按下后松开
            {
                erase_eeprom(add);               //擦除原有数据
                store_eeprom(add,speed);         //保存电机速度
                store_eeprom(add+1,time);        //保存运行时间
            }
        }
        else                         //状态标志位为 1
        {
        }
    }
}

void init_sys()                      //系统初始化函数
{
    speed=read_eeprom(add);          //读取电机速度
    if(speed==0xff)                  //没有保存数据
    {
        speed=50;
        time=3;                      //使用初始数值
    }
    else                             //存有保存数据
    {
        speed=read_eeprom(add);      //读取电机速度
        time=read_eeprom(add+1);     //读取运行时间
    }
    ...
}
```

4.2.3 step3：电梯运行模块设计

本步骤主要完成的具体任务是检测人员进入和离开电梯，搭乘人数为 1~10

人时按下启动按键控制电梯运行，搭乘人数超过 10 人时蜂鸣器报警 1s 提醒超载，其中定时功能使用定时器 T0 实现。

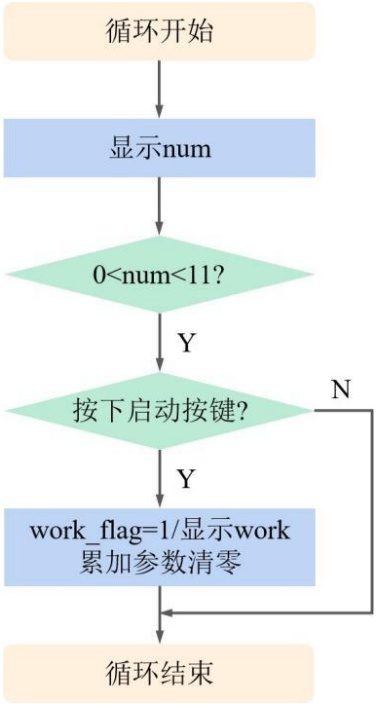
分析具体任务中与定时相关的控制功能，定时时间包括定时 1s 和定时 time，定时 1s 后蜂鸣器停止，定时 time 后电机停止并结束时间周期重新开始定时，系统输出包括电机输出和蜂鸣器输出，电机输出需要旋转和停止相应时长进行调速控制，蜂鸣器输出需要每间隔一定时间进行一次取反操作。因此，设定定时器 T0 的定时时间为 100μs，累加至 100 为定时 10ms，累加至 100*100 为定时 1s，累加至 100*100*time 为定时 time，定时时间结束置相应标志位。设定直流电机的工作周期为 10ms，按照 100%占空比划分单元时间为 100μs，通过比较累加参数和电机速度进行电机输出。设定蜂鸣器的发声频率为 10000Hz，对应周期为 100μs，通过每 100μs 进入定时中断取反蜂鸣器端口进行蜂鸣器输出。

系统主程序需要实现启动按键和液晶显示相关的控制功能，程序设计流程图如图 6.5(a)所示。通过按下启动按键控制电机输出，同时实时刷新运行状态和搭乘人数，设定 bit 类型的运行标志位 work_flag，当 work_flag=0 时为电机停止状态，当 work_flag=1 时为电机工作状态。外部中断 0 和外部中断 1 需要实现搭乘人数相关的控制功能，程序设计流程图如图 6.5(b)所示。其中外部中断 0 对人员进入信号计数，超过 10 人则控制蜂鸣器输出，外部中断 1 对人员离开信号计数，设定 bit 类型的发声标志位 sound_flag，当 sound_flag=0 时为蜂鸣器停止状态，当 sound_flag=1 时为蜂鸣器发声状态。定时器 T0 需要实现与定时相关的控制功能，程序设计流程图如图 6.5(c)所示，分为定时累加部分（time）、电机输出部分（motor）和蜂鸣器输出部分（sound），其中 count、count_10ms、count_1s 为定时累加参数。需要注意的是，当定时器持续工作时，定时开始需要清零一系列累加参数以得到精确的定时时间。系统标志位对应关系如表 6.8 所示。

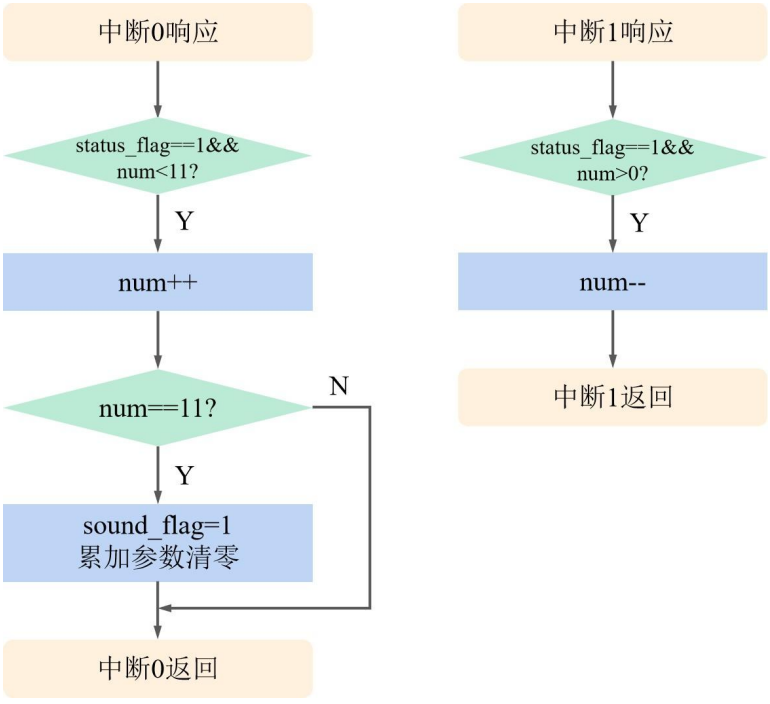
表 6.8 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
work_flag	0	电机停止
	1	电机工作
sound_flag	0	蜂鸣器停止
	1	蜂鸣器发声

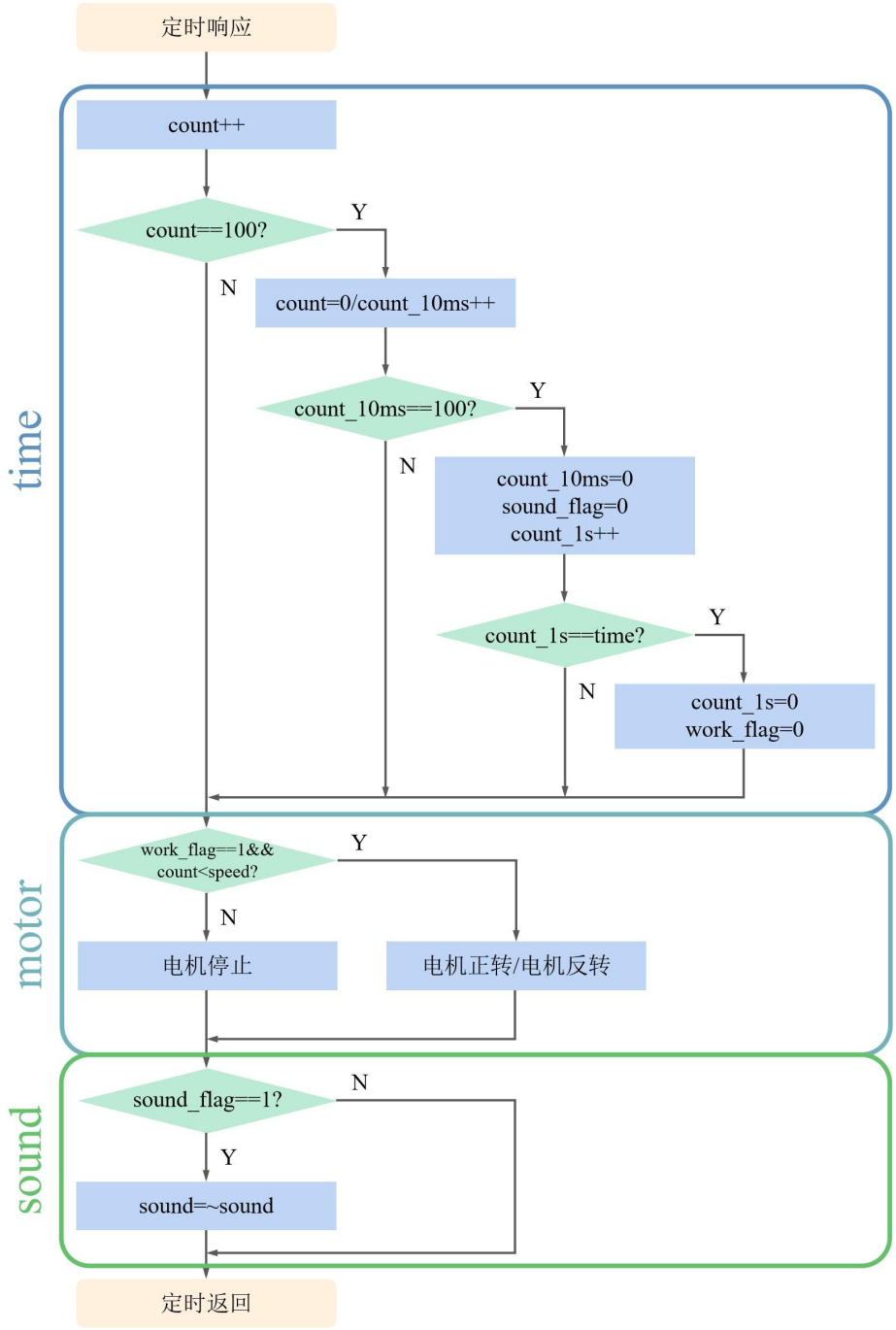
针对本步骤的设计任务，设计思路应首先考虑实现定时器 T0 部分，进行定时初始化配置并建立中断服务程序，定时累加置标志位同时执行相关系统输出，然后在此基础上实现外部中断 0 和外部中断 1 部分，进行中断初始化配置并建立中断服务程序，分别对人员进入和离开信号计数，最后实现系统主程序部分，检测启动按键是否按下并置运行标志位，同时刷新显示搭乘人数和运行状态。



(a)



(b)



(c)

图 6.5 电梯运行模块程序设计流程图

（1）首先完成定时器 T0 部分。进行定时初始化配置，包括设置寄存器 TMOD、设置定时器 T0 初值、打开中断允许开关、启动定时器 T0：设置定时器 T0 为方式 2 定时，则寄存器 TMOD 为 0x02；设置定时器 T0 的定时初值 TL0、TH0 为 0xA4，所涉及的定时初值可以在 STC 软件的定时器计算器中得到；打开总中断允许开关和定时器 T0 开关，并通过控制位 TR0 启动定时器 T0。根据电梯运行

模块程序设计流程图建立定时器 T0 函数。相应程序如下：

```
...
sbit sound=P3^6;                    //蜂鸣器端口 P3.6 定义
bit work_flag;                      //运行标志位定义
bit sound_flag;                     //发声标志位定义
uchar count,count_10ms,count_1s;    //定时累加参数定义

void init_sys()                     //系统初始化函数
{
    EA=1;                           //打开总中断允许开关
    ET0=1;                          //打开定时器 T0 开关
    TMOD=0x02;                      //设置定时器 T0 为方式 2 定时
    TL0=0xA4;
    TH0=0xA4;                       //设置定时器 T0 初值
    TR0=1;                          //启动定时器 T0
    ...
}

void timer0() interrupt 1           //定时器 T0 函数
{
    count++;                        //定时累加
    if(count==100)                  //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==100)         //定时 1s
        {
            count_10ms=0;
            sound_flag=0;           //置发声标志位为 0
            count_1s++;
            if(count_1s==time)      //定时 time
            {
                count_1s=0;
                work_flag=0;        //置运行标志位为 0
            }
        }
    }
    if(work_flag==1&&count<speed) //运行标志位为 1 且 count<speed
    {
        if(motor_flag==0)           //转向标志位为 0
        {
            fwd();                  //电机正转
        }
        else                        //转向标志位为 1
```

```
        {
            back();                     //电机反转
        }
    }
    else                                //运行标志位为 0 或 count>speed
    {
        stop();                         //电机停止
    }
    if(sound_flag==1)                   //发声标志位为 1
    {
        sound=~sound;                   //蜂鸣器端口取反
    }
}
```

（2）然后完成外部中断 0 和外部中断 1 部分。进行中断初始化配置，打开外部中断 0 和外部中断 1 开关并选择中断触发方式，根据电梯运行模块程序设计流程图建立外部中断 0 函数和外部中断 1 函数。相应程序如下：

```
void init_sys()                         //系统初始化函数
{
    EA=1;                               //打开总中断允许开关
    EX0=1;                              //打开外部中断 0 开关
    IT0=1;                              //选择跳沿触发方式
    EX1=1;                              //打开外部中断 1 开关
    IT1=1;                              //选择跳沿触发方式
    ...
}

void int_ex0() interrupt 0              //外部中断 0 函数
{
    if(status_flag==1&&num<11)          //状态标志位为 1 且人数不到上限
    {
        num++;                          //人员进入计数
        if(num==11)                     //搭乘人数超过 10 人
        {
            sound_flag=1;               //置发声标志位为 1
            count=count_10ms=count_1s=0;        //定时累加参数清零
        }
    }
}

void int_ex1() interrupt 2              //外部中断 1 函数
{
    if(status_flag==1&&num>0)           //状态标志位为 1 且人数不到下限
    {
```

```
            num--;                          //人员离开计数
        }
    }
```

（3）最后完成系统主程序部分，定义所需的显示内容为 code 型字符串数组。当启动按键按下后松开，置运行标志位为 1 并清零累加参数，此时电机工作重新定时至运行时间后自动停止。此外，搭乘人数 num 需要实时刷新放置在外层执行，运行状态 work 能够跟随启动按键刷新显示，运行状态 stop 则根据运行标志位刷新显示。相应程序如下：

```
...
uchar code disp_work[]  ={"work"};              //运行状态 work 定义

void main()                         //主函数
{
    init_sys();                     //系统初始化
    while(1)
    {
        ...
        key_last=key_now;
        key_now=scan_key();         //矩阵按键键值刷新
        if(status_flag==0)          //状态标志位为 0
        {
        }
        else                        //状态标志位为 1
        {
            lcd_pos(2,14);          //显示位置第 2 行第 14 列
            lcdwrite_sz(num/10);
            lcdwrite_sz(num%10);    //显示搭乘人数
            if(num>0&&num<11)       //搭乘人数为 1~10 人
            {
                if(key_last==0x40&&key_now==0xff)
                                    //启动按键按下后松开
                {
                    work_flag=1;                //置运行标志位为 1
                    count=count_10ms=count_1s=0;//定时累加参数清零
                    lcd_pos(2,2);               //显示位置第 2 行第 2 列
                    lcdwrite_string(disp_work); //显示运行状态 work
                }
                if(work_flag==0)    //运行标志位为 0
                {
                    lcd_pos(2,2);               //显示位置第 2 行第 2 列
                    lcdwrite_string(disp_stop); //显示运行状态 stop
                }
```

```
            }
        }
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"motor.h"
#include"1602LCD.h"
#include"matrix_key.h"
#include"stc89c52_eeprom.h"
#define uchar unsigned char
#define uint unsigned int
uchar code name[] ={"        TIANTI        "};      //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};   //欢迎标语 disp 定义
uchar code disp_fwd[]    ={"fwd "};                 //电机转向 fwd 定义
uchar code disp_back[]   ={"back"};                 //电机转向 back 定义
uchar code disp_speed[]  ={" S:"};                  //电机速度 speed 定义
uchar code disp_time[]   ={" T:"};                  //运行时间 time 定义
uchar code disp_stop[]   ={"stop"};                 //运行状态 stop 定义
uchar code disp_work[]   ={"work"};                 //运行状态 work 定义
uchar code disp_num[]    ={" number:"};             //搭乘人数 num 定义
sbit sound=P3^6;                        //蜂鸣器端口 P3.6 定义
sbit K4=P3^5;                           //功能按键端口 P3.5 定义
bit K4_last=1,K4_now=1;                 //功能按键键值初始化
uchar key_last=0xff,key_now=0xff;       //矩阵按键键值初始化
bit status_flag;                        //状态标志位定义
bit motor_flag;                         //转向标志位定义
bit speed_flag;                         //速度标志位定义
bit time_flag;                          //时间标志位定义
bit work_flag;                          //运行标志位定义
bit sound_flag;                         //发声标志位定义
uchar speed=50;                         //电机速度定义
uchar time=3;                           //运行时间定义
uchar num=0;                            //搭乘人数定义
uchar set_speed[2];                     //输入速度数组定义
uchar set_time;                         //输入时间定义
uchar count,count_10ms,count_1s;        //定时累加参数定义
uint add=0x2023;                        //EEPROM 地址定义
void init_sys();                        //系统初始化函数声明
```

```
void Delay1000ms();                     //延时 1000ms 函数声明

void main()                             //主函数
{
    init_sys();                         //系统初始化
    while(1)
    {
        K4_last=K4_now;
        K4_now =K4;                     //功能按键键值刷新
        if(K4_last==0&&K4_now==1&&work_flag==0)
                                        //电机停止状态下功能按键按下后松开
        {
            status_flag=~status_flag;   //状态标志位取反
            if(status_flag==0)          //状态标志位为 0
            {
                lcd_pos(1,1);           //显示位置第 1 行第 1 列
                lcdwrite_zm('*');       //显示符号*
                lcd_pos(2,1);
                lcdwrite_zm(' ');
            }
            else                        //状态标志位为 1
            {
                lcd_pos(1,1);
                lcdwrite_zm(' ');
                lcd_pos(2,1);           //显示位置第 2 行第 1 列
                lcdwrite_zm('*');       //显示符号*
            }
        }
        key_last=key_now;
        key_now=scan_key();             //矩阵按键键值刷新
        if(status_flag==0)              //状态标志位为 0
        {
            if(key_last==0x10&&key_now==0xff)  //转向设置按键按下后松开
            {
                motor_flag=~motor_flag;         //转向标志位取反
                if(motor_flag==0)               //转向标志位为 0
                {
                    lcd_pos(1,2);               //显示位置第 1 行第 2 列
                    lcdwrite_string(disp_fwd);  //显示电机转向 fwd
                }
                else                            //转向标志为为 1
                {
                    lcd_pos(1,2);               //显示位置第 1 行第 2 列
                    lcdwrite_string(disp_back); //显示电机转向 back
```

```
    }
}
else if(key_last==0x20&&key_now==0xff&&time_flag==0)
                            //速度设置按键按下后松开
{
    speed_flag=1;           //置速度标志位为 1
    lcd_pos(1,8);           //显示位置第 1 行第 8 列
    lcdwrite_zm('(');
    lcdwrite_zm('_');
    lcdwrite_zm('_');
    lcdwrite_zm(')');       //电机速度位置刷新显示
}
else if(key_last==0x30&&key_now==0xff&&speed_flag==0)
                            //时间设置按键按下后松开
{
    time_flag=1;            //置时间标志位为 1
    lcd_pos(1,13);          //显示位置第 1 行第 13 列
    lcdwrite_zm('(');
    lcdwrite_zm('_');
    lcdwrite_zm(')');       //运行时间位置刷新显示
}
else if(key_last==0x50&&key_now==0xff)
                            //保存按键按下后松开
{
    erase_eeprom(add);                  //擦除原有数据
    store_eeprom(add,speed);            //保存电机速度
    store_eeprom(add+1,time);           //保存运行时间
}
else if(key_last==0x60&&key_now==0xff)
                            //确定按键按下后松开
{
    if(speed_flag==1)       //速度标志位为 1
    {
        speed_flag=0;       //置速度标志位为 0
        if(set_speed[1]+set_speed[0]!=0)//检查速度设置范围
        {
            speed=set_speed[1]*10+set_speed[0];
        }
        lcd_pos(1,6);       //显示位置第 1 行第 6 列
        lcdwrite_string(disp_speed);
        lcdwrite_sz(speed/10);
        lcdwrite_sz(speed%10);
        lcdwrite_zm(' ');   //电机速度位置刷新显示
    }
```

```
            if(time_flag==1)        //时间标志位为 1
            {
                time_flag=0;        //置时间标志位为 0
                if(set_time!=0)     //检查时间设置范围
                {
                    time=set_time;
                }
                lcd_pos(1,11);      //显示位置第 1 行第 11 列
                lcdwrite_string(disp_time);
                lcdwrite_sz(time);
                lcdwrite_zm(' ');   //运行时间位置刷新显示
            }
        }
        if(speed_flag==1)           //速度标志位为 1
        {
            if(key_last<=9&&key_now==0xff)  //0~9 数字键按下后松开
            {
                set_speed[1]=set_speed[0];
                set_speed[0]=key_last;          //保存输入速度
                lcd_pos(1,9);                   //显示位置第 1 行第 9 列
                lcdwrite_sz(set_speed[1]);
                lcdwrite_sz(set_speed[0]);      //显示输入速度
            }
        }
        if(time_flag==1)            //时间标志位为 1
        {
            if(key_last<=9&&key_now==0xff)  //0~9 数字键按下后松开
            {
                set_time=key_last;              //保存输入时间
                lcd_pos(1,14);                  //显示位置第 1 行第 14 列
                lcdwrite_sz(set_time);          //显示输入时间
            }
        }
    }
    else                            //状态标志位为 1
    {
        lcd_pos(2,14);              //显示位置第 2 行第 14 列
        lcdwrite_sz(num/10);
        lcdwrite_sz(num%10);        //显示搭乘人数
        if(num>0&&num<11)           //搭乘人数为 1~10 人
        {
            if(key_last==0x40&&key_now==0xff)
                                    //启动按键按下后松开
            {
```

```
                    work_flag=1;                    //置运行标志位为 1
                    count=count_10ms=count_1s=0;//定时累加参数清零
                    lcd_pos(2,2);                   //显示位置第 2 行第 2 列
                    lcdwrite_string(disp_work);     //显示运行状态 work
                }
                if(work_flag==0)        //运行标志位为 0
                {
                    lcd_pos(2,2);                   //显示位置第 2 行第 2 列
                    lcdwrite_string(disp_stop);     //显示运行状态 stop
                }
            }
        }
    }
}

void init_sys()                         //系统初始化函数
{
    EA=1;                               //打开总中断允许开关
    ET0=1;                              //打开定时器 T0 开关
    EX0=1;                              //打开外部中断 0 开关
    IT0=1;                              //选择跳沿触发方式
    EX1=1;                              //打开外部中断 1 开关
    IT1=1;                              //选择跳沿触发方式
    TMOD=0x02;                          //设置定时器 T0 为方式 2 定时
    TL0=0xA4;
    TH0=0xA4;                           //设置定时器 T0 初值
    TR0=1;                              //启动定时器 T0
    speed=read_eeprom(add);             //读取电机速度
    if(speed==0xff)                     //没有保存数据
    {
        speed=50;
        time=3;                         //使用初始数值
    }
    else                                //存有保存数据
    {
        speed=read_eeprom(add);         //读取电机速度
        time=read_eeprom(add+1);        //读取运行时间
    }
    lcd_init();                         //液晶显示初始化
    lcd_pos(1,1);                       //显示位置第 1 行第 1 列
    lcdwrite_string(name);              //显示系统名称 name
    lcd_pos(2,1);                       //显示位置第 2 行第 1 列
    lcdwrite_string(disp);              //显示欢迎标语 disp
    Delay1000ms();                      //延时 1000ms
```

```
    lcd_clear();                    //液晶显示清屏
    lcd_pos(1,1);                   //显示位置第 1 行第 1 列
    lcdwrite_zm('*');               //显示符号*
    lcdwrite_string(disp_fwd);      //显示电机转向 fwd
    lcdwrite_string(disp_speed);    //显示电机速度 speed
    lcdwrite_sz(speed/10);
    lcdwrite_sz(speed%10);          //显示电机速度
    lcdwrite_string(disp_time);     //显示运行时间 time
    lcdwrite_sz(time);              //显示运行时间
    lcd_pos(2,1);                   //显示位置第 2 行第 1 列
    lcdwrite_zm(' ');               //显示空格
    lcdwrite_string(disp_stop);     //显示运行状态 stop
    lcdwrite_string(disp_num);      //显示搭乘人数 num
    lcdwrite_sz(num/10);
    lcdwrite_sz(num%10);            //显示搭乘人数
}

void int_ex0() interrupt 0          //外部中断 0 函数
{
    if(status_flag==1&&num<11)      //状态标志位为 1 且人数不到上限
    {
        num++;                      //人员进入计数
        if(num==11)                 //搭乘人数超过 10 人
        {
            sound_flag=1;           //置发声标志位为 1
            count=count_10ms=count_1s=0;        //定时累加参数清零
        }
    }
}

void int_ex1() interrupt 2          //外部中断 1 函数
{
    if(status_flag==1&&num>0)       //状态标志位为 1 且人数不到下限
    {
        num--;                      //人员离开计数
    }
}

void timer0() interrupt 1           //定时器 T0 函数
{
    count++;                        //定时累加
    if(count==100)                  //定时 10ms
    {
        count=0;
```

```
            count_10ms++;
            if(count_10ms==100)          //定时 1s
            {
                count_10ms=0;
                sound_flag=0;            //置发声标志位为 0
                count_1s++;
                if(count_1s==time)        //定时 time
                {
                    count_1s=0;
                    work_flag=0;         //置运行标志位为 0
                }
            }
        }
        if(work_flag==1&&count<speed) //运行标志位为 1 且 count<speed
        {
            if(motor_flag==0)            //转向标志位为 0
            {
                fwd();                   //电机正转
            }
            else                         //转向标志位为 1
            {
                back();                  //电机反转
            }
        }
        else                             //运行标志位为 0 或 count>speed
        {
            stop();                      //电机停止
        }
        if(sound_flag==1)                //发声标志位为 1
        {
            sound=~sound;                //蜂鸣器端口取反
        }
    }

    void Delay1000ms()                   //延时 1000ms 函数
    {
        unsigned char i, j, k;
        _nop_();
        i = 8;
        j = 1;
        k = 243;
        do
        {
            do
```

```
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

5.本章小结

本章设计开发了一种太空电梯控制系统，结合前面章节所学知识实现太空电梯的参数设置和电梯运行，使用矩阵按键作为交互模块，极大地扩展了按键数目，丰富了按键功能，使用 EEPROM 保存关键数据，通过系统断电开机后重新读取数据实现断电保持功能。同时，本章进一步深化了直流电机、独立按键、液晶显示、外部中断、定时器的综合应用，其中使用液晶显示需要注意刷新显示的覆盖情况，同时根据程序设计的具体需求安排液晶显示的调用位置，使用定时器则需要注意定时开始清零一系列累加参数以得到精确的定时时间。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用矩阵按键和 EEPROM 的控制功能，并能够应用前面章节所学知识独立完成综合控制系统的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，要求实现太空电梯和远程电脑之间的双向通信，一方面电梯可以传输数据至电脑，另一方面电脑也可以进行参数设置并控制电梯运行，具体功能自拟。

一种亚运会赛事售票控制系统开发

1.学习目标

1. 能根据控制系统任务要求基于单片机开发板硬件分步完成软件程序设计；
2. 能应用串口通信实现单片机和个人计算机之间的双向通信；
3. 能结合任务实际情况进行控制系统方案优化设计。

2.任务导入

随着计算机网络技术的普及，单片机可以和计算机相结合，通过串口通信进行数据传输和信息交换，从而实现数据采集、过程控制、远程监控等多种控制功能。AT89C52 单片机包含 1 个异步收发串口通信接口，使用 2 个独立缓冲器 SBUF 接收和发送数据，同时通过寄存器 SCON 选择工作方式控制串口通信。此外，接收和发送结束时会置相应标志位产生中断请求，如果中断请求被允许的话，单片机将会响应该中断请求并自动跳转执行中断服务程序。本章提出一例“设计一种亚运会赛事售票控制系统”的课题需求，学习串口通信的初始设置和使用方法，并应用前面章节所学知识完成控制系统的设计工作。

任务题目：设计一种亚运会赛事售票控制系统

功能要求：该系统将单片机作为后台处理设备，将计算机作为用户交互平台，通过单片机和计算机之间的双向通信实现用户依次进行赛事选择、座位选择、订单确认的一系列操作。其中串口通信选择 8 位异步收发且波特率可变的工作方式，并使用定时器 T1 作为波特率发生器。

具体控制要求如下：

1.初始界面显示。系统上电后在 1602LCD 第 1 行显示系统名称“19th Asian Games”，第 2 行显示欢迎标语“U ARE WELLCOME”。1 秒后清屏，进入赛事选择界面。

2.赛事选择功能。

赛事选择界面：第 1 行显示界面名称“Competition List”，第 2 行滚动显示赛事名称，分别为“1.Athletics”（田径）、“2.Gymnastics”（体操）、“3.Basketball”（篮球）、“4.Swimming”（游泳）。

赛事选择功能：用户在计算机上输入并发送所选赛事的对应数字，单片机接收到数字信号后自动进入该项赛事的座位选择界面。

3.座位选择功能。

座位选择界面：第 1 行显示赛事名称，第 2 行依次显示座位排数、座位号码、门票价格信息“row00num00RMB000”，其中座位排数和座位号码的选择范围均为 1~99，门票价格则随着座位排数的增加而减少，具体关系为门票价格=1000-5*座位排数。

座位选择功能：用户在计算机上输入并发送 4 位数字给单片机，其中第 1 位数字和第 2 位数字为座位排数，第 3 位数字和第 4 位数字为座位号码。如果座位排数和座位号码均为有效，则单片机刷新显示数值并返回门票价格，然后自动进入订单确认界面。

4.订单确认功能。

订单确认界面：第 1 行显示确认信息“Confirm:1yes 2no”，第 2 行仍然显示座位排数、座位号码、门票价格信息。

订单确认功能：用户在计算机上输入并发送确认信息的对应数字，如果发送数字“1”，则继电器导通出票，2 秒后自动断开，返回出票信息“Ticket Issued!”并自动回到赛事选择界面，如果发送数字“2”，则直接回到赛事选择界面重新进行赛事选择和座位选择。

3.控制系统方案设计

根据任务功能要求，基于单片机开发板硬件，分步完成软件程序设计，从而进行控制系统开发。其中控制系统硬件设计包括两部分：一是单片机 I/O 接口电路及模块选型，二是单片机内部硬件资源配置。

3.1 单片机输出模块

单片机输出模块包括继电器控制模块和液晶显示控制模块。

（1）继电器控制模块主要用于订单确认功能的出票装置控制。单片机输出信号控制继电器线圈得电吸合或断电释放，其控制原理已在第三章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P3.7 引脚。

（2）液晶显示控制模块主要用于系统运行过程中各项信息数据的实时刷新显示。单片机通过输出数据和命令至 1602LCD，从而实现相应的数字、字母、符号显示，其控制原理已在第三章案例中详细讲解，此处不作赘述。在开发板上，该模块占用 P0+P2.0/P2.1/P2.2 引脚。

3.2 单片机内部配置

单片机内部配置包括定时器配置和串口通信配置。

（1）定时器配置主要用于系统运行过程中的时间控制，其中定时器 T0 选择定时模式，实现定时相关的控制功能，定时器 T1 同样选择定时模式，作为串口通信的波特率发生器。

（2）串口通信配置主要用于单片机和个人计算机之间的数据传输和信息交换。该模块通过 2 个数据引脚接收和发送数据，其内部结构如图 7.1 所示，在开发板上占用 P3.0/P3.1 引脚，其中 RXD（P3.0）引脚用于接收数据，TXD（P3.1）引脚用于发送数据。

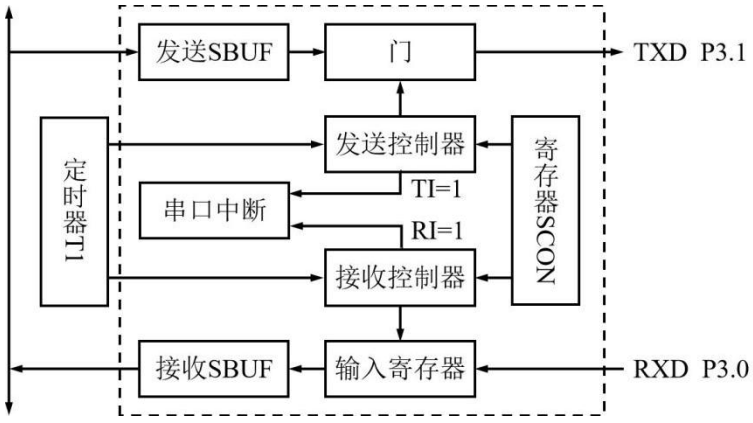


图 7.1 串口通信内部结构图

系统端口及硬件配置如表 7.1 所示。

表 7.1 系统端口及硬件配置表

序号	系统端口/内部资源	功能配置
1	P3.7	继电器
2	P0+P2.0/P2.1/P2.2	液晶显示
3	定时器 T0	定时功能
4	定时器 T1	串口通信的波特率发生器
5	串口通信	单片机和计算机的双向通信

4.控制系统软件设计

在控制系统硬件设计的基础上进行软件设计。首先确定系统软件架构，然后根据架构进行 step by step 程序设计，绘制程序流程图，分步细化完成具体的程序编写，最后整合成完整的系统软件程序。

4.1 系统软件架构

分析控制系统的功能要求，可以将其罗列为“赛事选择”、“座位选择”、“订单确认”三个同一级别的模块，设定 unsigned char 类型的状态标志位 status_flag。系统上电后默认 status_flag=0 进入“赛事选择”模块，此时可执行赛事选择操作，执行结束置 status_flag=1 进入“座位选择”模块，此时可执行座位选择操作，执行结束置 status_flag=2 进入“订单确认”模块，此时可执行订单确认操作，执行结束置 status_flag=0 回到“赛事选择”模块。

系统标志位对应关系如表 7.2 所示。

表 7.2 系统标志位对应关系表

标志位名称	标志位数值	对应当前状态
status_flag	0	赛事选择
	1	座位选择
	2	订单确认

系统软件架构如图 7.2 所示。

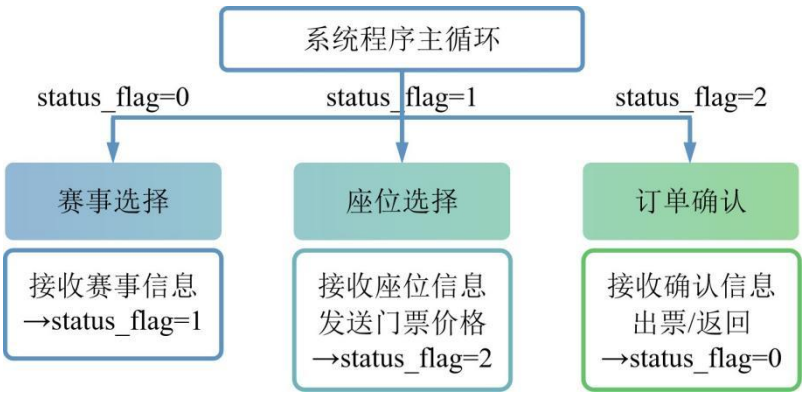


图 7.2 系统软件架构

4.2 step by step 程序设计

将软件设计的总体任务分解为由四个关键技术点关联的程序设计任务，并在系统软件架构的基础上拟定程序设计步骤，然后以“搭积木”的思路开展程序设计实践。系统关键技术点如表 7.3 所示。

表 7.3 step by step 程序设计及其关键技术点

序号	技术点	关键点
Step 1	系统程序框架设计	标志位划分系统状态的应用
Step 2	赛事选择模块设计	串口通信的应用
Step 3	座位选择模块设计	串口通信的应用
Step 4	订单确认模块设计	串口通信的应用

4.2.1 step1：系统程序框架设计

在进行控制系统各个模块的程序设计之前，搭建系统程序框架，即将软件架构以程序语言的形式体现。此外，系统上电后需要进行初始界面显示，放置在主函数的 while(1)循环前执行。

针对本步骤的设计任务，根据软件架构使用标志位划分系统状态以执行不同模块程序即可，同时根据功能要求调用液晶显示函数实现初始界面显示。

（1）在程序开头对 8051 单片机所需的头文件进行 include 包含处理，其中“1602LCD.h”为液晶显示头文件。定义 status_flag 为状态标志位：当状态标志

位为 0 时，执行“赛事选择”模块程序；当状态标志位为 1 时，执行“座位选择”模块程序；当状态标志位为 2 时，执行“订单确认”模块程序。相应程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"1602LCD.h"
#define uchar unsigned char
#define uint unsigned int
uchar status_flag;                    //状态标志位定义

void main()                           //主函数
{
    while(1)
    {
        if(status_flag==0)            //状态标志位为 0
        {
        }
        else if(status_flag==1)       //状态标志位为 1
        {
        }
        else if(status_flag==2)       //状态标志位为 2
        {
        }
    }
}
```

（2）定义初始界面显示所需的系统名称和欢迎标语为 code 型字符串数组：首先进行液晶显示初始化，指定第 1 行第 1 列显示系统名称 name，第 2 行第 1 列显示欢迎标语 disp，然后延时 1000ms 进行液晶显示清屏，进入“赛事选择”模块，所涉及的延时函数可以在 STC 软件的软件延时计算器中得到。相应程序如下：

```
...
uchar code name[] ={"19th Asian Games"};        //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};   //欢迎标语 disp 定义
void Delay1000ms();                   //延时 1000ms 函数声明

void main()                           //主函数
{
    lcd_init();                       //液晶显示初始化
    lcd_pos(1,1);                     //显示位置第 1 行第 1 列
    lcdwrite_string(name);            //显示系统名称 name
    lcd_pos(2,1);                     //显示位置第 2 行第 1 列
    lcdwrite_string(disp);            //显示欢迎标语 disp
```

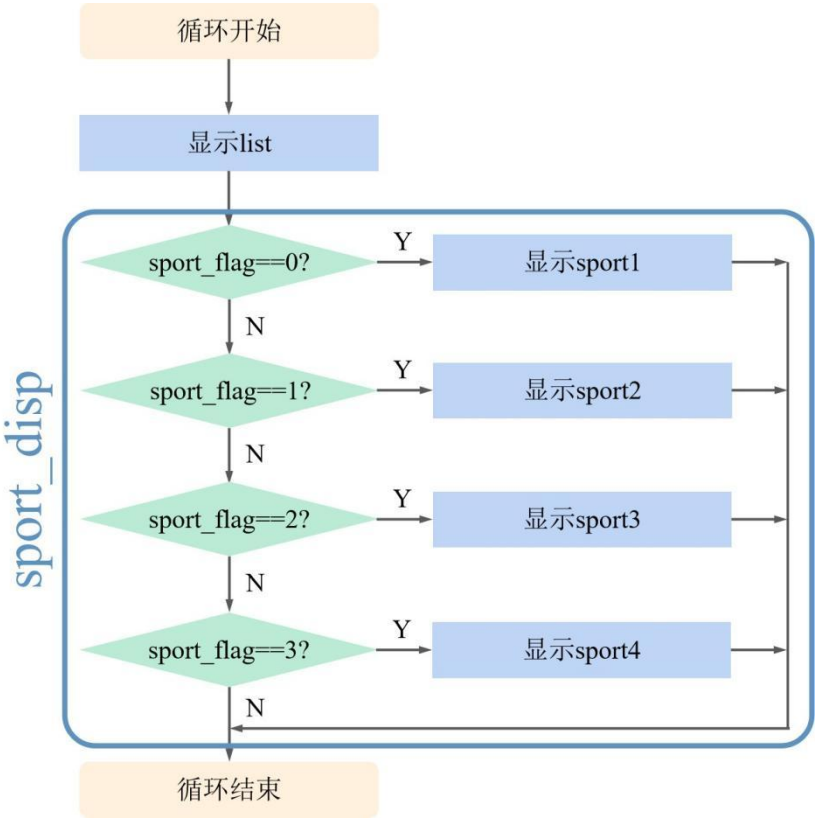
```
    Delay1000ms();                  //延时 1000ms
    lcd_clear();                    //液晶显示清屏
    while(1)
    {
    }
}

void Delay1000ms()                  //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

4.2.2 step2：赛事选择模块设计

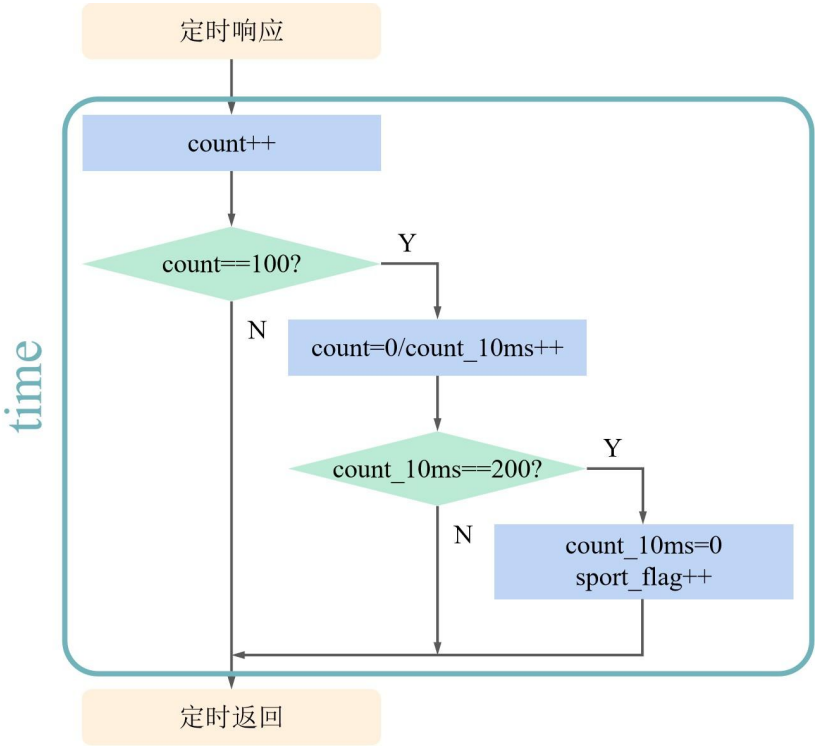
本步骤主要完成的具体任务是单片机以一定频率滚动显示赛事选择界面，接收到数字信号后自动进入座位选择界面。其中与串口通信无关的控制功能为滚动显示赛事选择界面，与串口通信相关的控制功能为单片机接收 1 位数字。

首先分析具体任务中与串口通信无关的控制功能。根据功能要求调用液晶显示函数实现赛事界面显示，同时通过间隔定时时间刷新赛事名称实现赛事滚动显示。因此，设定滚动显示的定时时间为 2s，定时器 T0 的定时时间为 100μs，累加至 100 为定时 10ms，累加至 100*200 为定时 2s，定时时间结束置相应标志位并结束时间周期重新开始定时。

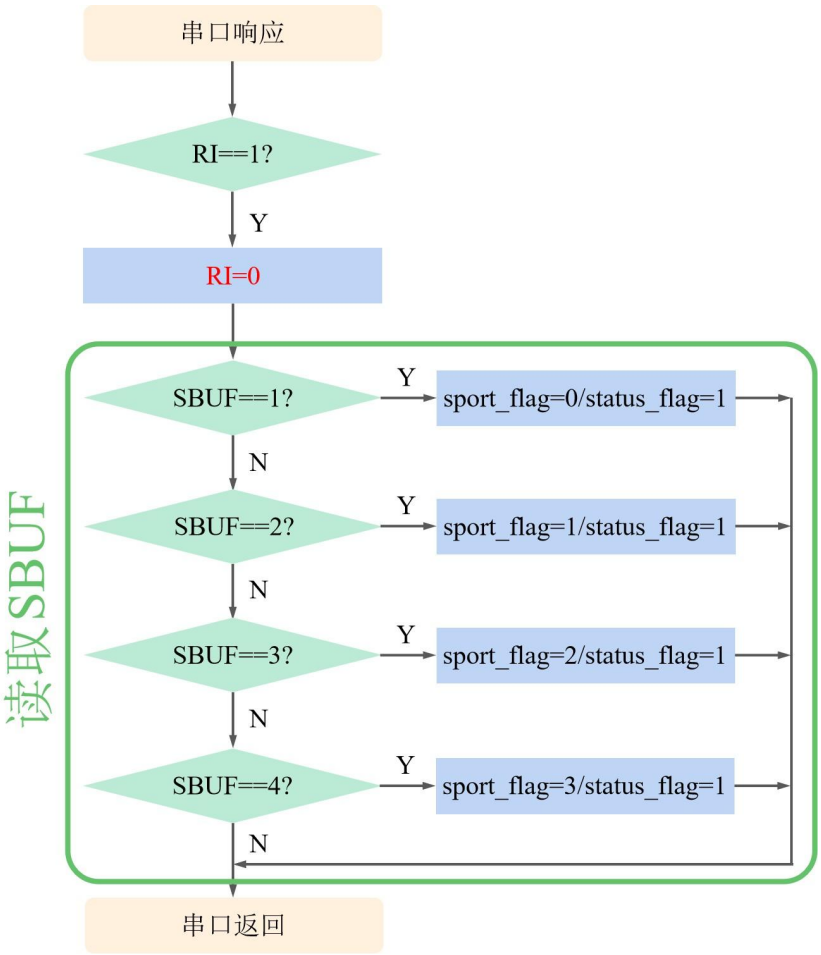
然后分析具体任务中与串口通信相关的控制功能。计算机发送数据存入接收缓冲器 SBUF，单片机通过读取 SBUF 接收数据，然后自动进入座位选择界面。因此，单片机接收数据结束需要进入串口中断，判断接收数字并置相应标志位，此时主程序切换系统状态并刷新显示界面。


循环开始
显示list
sport_disp
sport_flag==0?
Y
显示sport1
N
sport_flag==1?
Y
显示sport2
N
sport_flag==2?
Y
显示sport3
N
sport_flag==3?
Y
显示sport4
N
循环结束

(a)


定时响应
time
count++
count==100?
Y
N
count=0/count_10ms++
count_10ms==200?
Y
N
count_10ms=0
sport_flag++
定时返回

(b)



(c)

图 7.3 赛事选择模块程序设计流程图

系统主程序需要实现液晶显示相关的控制功能，程序设计流程图如图 7.3(a)所示。设定 unsigned char 类型的赛事标志位 sport_flag，当 sport_flag=0 时为田径赛事，当 sport_flag=1 时为体操赛事，当 sport_flag=2 时为篮球赛事，当 sport_flag=3 时为游泳赛事。定时器 T0 需要实现与定时相关的控制功能，程序设计流程图如图 7.3(b)所示，其中 count、count_10ms 为定时累加参数。串口通信需要实现与接收数据相关的控制功能，程序设计流程图如图 7.3(c)所示。需要注意的是，单片机接收数据结束时会置接收标志位 RI=1，产生中断请求触发中断响应，同时必须由程序手动清零置 RI=0。

针对本步骤的设计任务，设计思路应首先考虑实现定时器 T0 部分，进行定时初始化配置并建立中断服务程序，定时参数累加并置赛事标志位，然后在此基础上实现系统主程序部分，根据赛事标志位刷新赛事名称，最后实现串口通信部分，进行串口初始化配置并建立中断服务程序，同时置赛事标志位和状态标志位。

（1）首先完成定时器 T0 部分。进行定时初始化配置，包括设置寄存器 TMOD、设置定时器 T0 初值、打开中断允许开关、启动定时器 T0：设置定时器 T0 为方式 2 定时，则寄存器 TMOD 为 0x02；设置定时器 T0 的定时初值 TL0、TH0 为 0xA4，所涉及的定时初值可以在 STC 软件的定时器计算器中得到；打开总中断允许开关和定时器 T0 开关，并通过控制位 TR0 启动定时器 T0。根据赛事选择模块程序设计流程图建立定时器 T0 函数。相应程序如下：

```
...
uchar sport_flag;                    //赛事标志位定义
uchar count,count_10ms;              //定时累加参数定义

void main()
{
    ...
    EA=1;                            //打开总中断允许开关
    ET0=1;                           //打开定时器 T0 开关
    TMOD=0x02;                       //设置定时器 T0 为方式 2 定时
    TL0=0xA4;
    TH0=0xA4;                        //设置定时器 T0 初值
    TR0=1;                           //启动定时器 T0
    while(1)
    {
    }
}

void timer0() interrupt 1            //定时器 T0 函数
{
    count++;                         //定时累加
    if(count==100)                   //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==200)          //定时 2s
        {
            count_10ms=0;
            sport_flag++;            //运动标志位累加
            if(sport_flag==4)        //运动标志位为 4
            {
                sport_flag=0;        //运动标志位清零
            }
        }
    }
```

}

（2）然后完成系统主程序部分。定义所需的显示内容为 code 型字符串数组，根据赛事选择模块程序设计流程图建立赛事显示函数。同时，在主函数的 while(1) 循环中指定第 1 行第 1 列显示界面名称 list，第 2 行第 1 列调用赛事显示函数显示赛事名称 sport。相应程序如下：

```
...
uchar code list[]    ={"Competition List"};          //界面名称 list 定义
uchar code sport1[]={"   1.Athletics     "};         //赛事名称 sport1 定义
uchar code sport2[]={"   2.Gymnastics    "};         //赛事名称 sport2 定义
uchar code sport3[]={"   3.Basketball    "};         //赛事名称 sport3 定义
uchar code sport4[]={"   4.Swimming       "};        //赛事名称 sport4 定义
void sport_disp();                          //赛事显示函数声明

void main()                                 //主函数
{
    ...
    while(1)
    {
        if(status_flag==0)                  //状态标志位为 0
        {
            lcd_pos(1,1);                   //显示位置第 1 行第 1 列
            lcdwrite_string(list);          //显示界面名称 list
            lcd_pos(2,1);                   //显示位置第 2 行第 1 列
            sport_disp();                   //显示赛事名称
        }
        else if(status_flag==1)             //状态标志位为 1
        {
        }
        else if(status_flag==2)             //状态标志位为 2
        {
        }
    }
}

void sport_disp()                           //赛事显示函数
{
    switch(sport_flag)
    {
        case 0:lcdwrite_string(sport1);break;       //标志位为 0 显示 sport1
        case 1:lcdwrite_string(sport2);break;       //标志位为 1 显示 sport2
        case 2:lcdwrite_string(sport3);break;       //标志位为 2 显示 sport3
        case 3:lcdwrite_string(sport4);break;       //标志位为 3 显示 sport4
```

```
    }
}
```

（3）最后完成串口通信部分。

使用串口通信需要进行串口初始化配置，包括设置寄存器 SCON 和打开中断允许开关，放置在主函数的 while(1)循环前执行。其中寄存器 SCON 用于选择串口通信的工作方式，其格式如图 7.4 所示。

	D7	D6	D5	D4	D3	D2	D1	D0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI

图 7.4 寄存器 SCON 格式

SM0、SM1 为工作方式选择位。根据 SM0、SM1 的不同设置，共有 4 种工作方式如表 7.4 所示。其中方式 0 为同步移位寄存器输入/输出方式，用于扩展并行 I/O 口，波特率为 FOSC/12；方式 1 为 8 位异步收发且波特率可变的工作方式，用于双机串行通信，波特率由定时器控制；方式 2 为 9 位异步收发且波特率固定的工作方式，用于多机串行通信，波特率为 FOSC/32 或 FOSC/64；方式 3 为 9 位异步收发且波特率可变的工作方式，用于多机串行通信，波特率由定时器控制。

表 7.4 SM0、SM1 工作方式

SM0	SM1	工作方式
0	0	方式 0，同步移位寄存器方式
0	1	方式 1，8 位异步收发且波特率可变
1	0	方式 2，9 位异步收发且波特率固定
1	1	方式 3，9 位异步收发且波特率可变

SM2 为多机通信控制位。当串口通信选择方式 2 和方式 3 时，SM2 用于控制将接收到的数据存入或丢弃。

REN 为允许串行接收位。当 REN=1 时，允许串行接口接收数据；当 REN=0 时，禁止串行接口接收数据。

TB8 为发送的第 9 位数据。当串口通信选择方式 2 和方式 3 时，TB8 用于存放发送的第 9 位数据。

RB8 为接收的第 9 位数据。当串口通信选择方式 2 和方式 3 时，RB8 用于存放接收的第 9 位数据。

TI 为发送中断标志位。当串口通信选择方式 0 时，发送数据结束时会置发

送标志位 TI=1，当串口通信选择其余方式时，发送数据停止位时会置发送标志位 TI=1，表示 1 帧数据发送结束。当 TI=1 时产生中断请求触发中断响应，同时必须由程序手动清零置 TI=0。

RI 为接收中断标志位。当串口通信选择方式 0 时，接收数据结束时会置接收标志位 RI=1，当串口通信选择其余方式时，接收数据停止位时会置接收标志位 RI=1，表示 1 帧数据接收结束。当 RI=1 时产生中断请求触发中断响应，同时必须由程序手动清零置 RI=0。

已知本例串口通信用于单片机和个人计算机之间的双向通信。因此，设置串口通信为方式 1 双机通信，同时置 REN=1 允许接收数据，此时寄存器 SCON 为 01010000，转换为十六进制为 0x50。此外，方式 1 的波特率由定时器控制，使用定时器 T1 作为波特率发生器，需要进行定时初始化配置，包括设置寄存器 TMOD、设置定时器 T1 初值、启动定时器 T1，同样放置在主函数的 while(1)循环前执行。所涉及的定时初值可以在 STC 软件的波特率计算器中得到，设置系统频率为 11.0592MHz，波特率为 9600，选择 UART 为串口 1，数据位为 8 位数据，如图 7.5 所示。

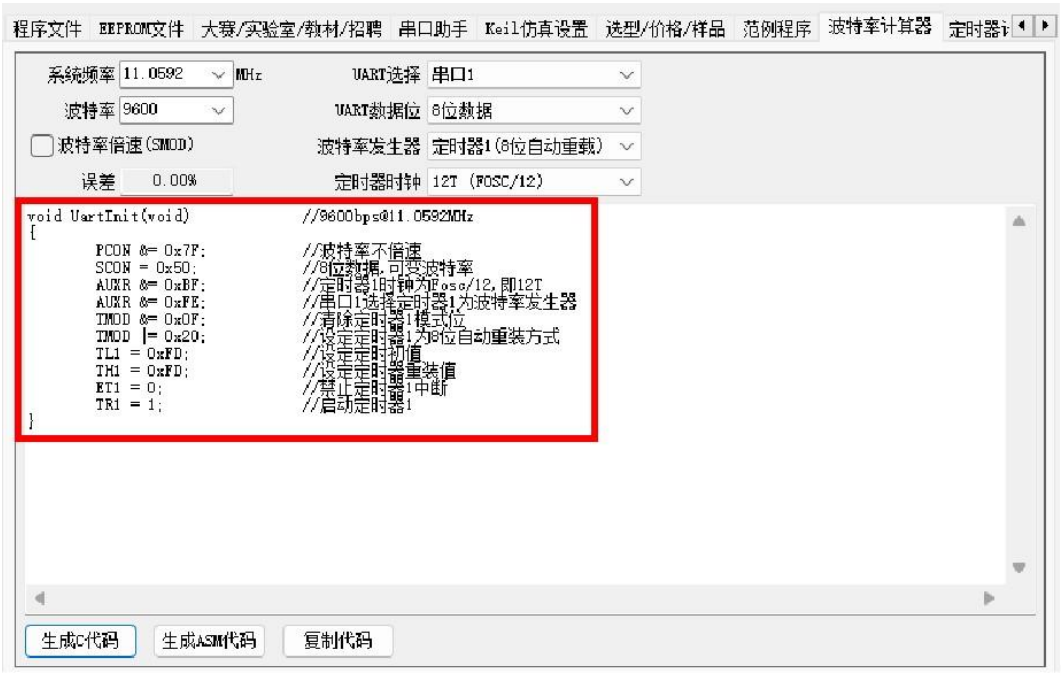


图 7.5 STC 波特率计算器界面

设置寄存器 SCON 之后，接收和发送结束时产生中断请求触发中断响应进行中断处理同样需要打开中断允许开关和建立中断服务程序。其中串口通信开关

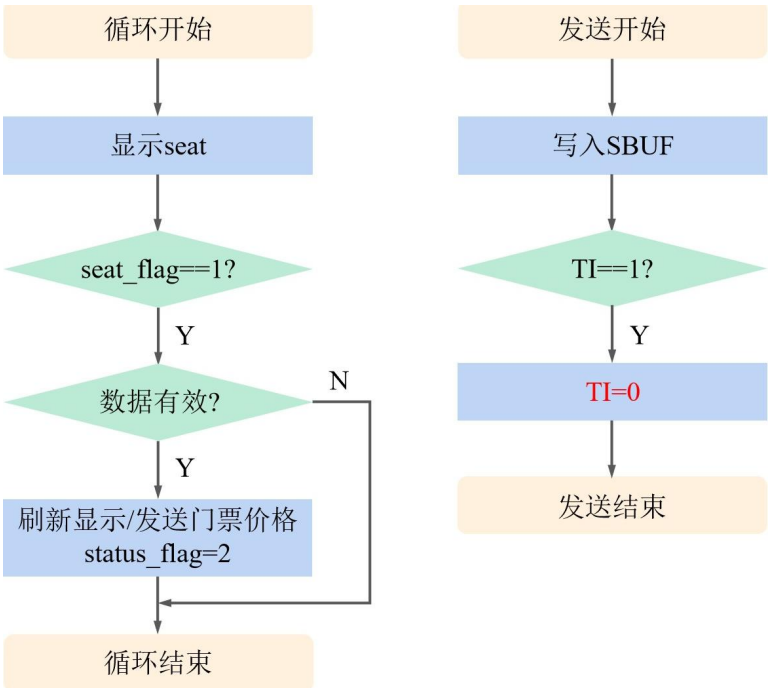
为 ES=1，串口通信的中断号为 4，可以建立中断服务程序为 void uart() interrupt 4。

根据赛事选择模块程序设计流程图进行串口初始化配置并建立串口通信函数。相应程序如下：

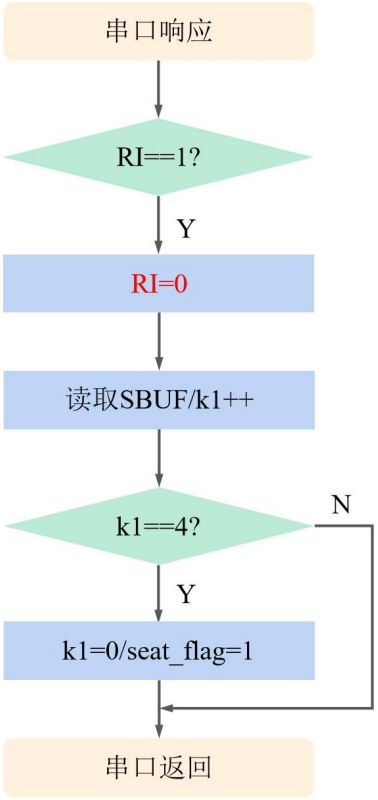
```
void main()
{
    ...
    EA=1;                          //打开总中断允许开关
    ES=1;                          //打开串口通信开关
    TMOD=0x52;                     //设置定时器 T0 为方式 2 定时
                                     设置定时器 T1 为方式 2 定时
    SCON=0x50;                     //设置串口通信为方式 1 且允许接收
    TL1=0xFD;
    TH1=0xFD;                      //设置定时器 T1 初值
    TR1=1;                         //启动定时器 T1
    while(1)
    {
    }
}

void uart() interrupt 4            //串口通信函数
{
    if(RI==1)                      //接收标志位为 1
    {
        RI=0;                      //接收标志位清零
        switch(SBUF)
        {
            case 1:sport_flag=0;status_flag=1;break;
            case 2:sport_flag=1;status_flag=1;break;
            case 3:sport_flag=2;status_flag=1;break;
            case 4:sport_flag=3;status_flag=1;break;
        }//SBUF 为 0/1/2/3 置赛事标志位为 1/2/3/4 并置状态标志位为 1
    }
}
```

4.2.3 step3：座位选择模块设计

本步骤主要完成的具体任务是单片机显示座位选择界面，接收到 4 位数字后如果数据有效则刷新显示数值并返回门票价格，然后自动进入订单确认界面。其中与串口通信无关的控制功能为显示座位选择界面，与串口通信相关的控制功能为单片机接收 4 位数字和单片机发送门票价格。



(a)



(b)

图 7.6 座位选择模块程序设计流程图

首先分析具体任务中与串口通信无关的控制功能，根据功能要求调用液晶显示函数实现座位界面显示即可。然后分析具体任务中与串口通信相关的控制功能。

单片机通过读取接收缓冲器 SBUF 接收数据，同时通过写入发送缓冲器 SBUF 发送数据。此外，串口通信以字节为单位进行数据传输，即单次收发过程仅为 1 位数字、字母、符号，通过多次接收和发送数据实现多位数据传输。因此，单片机需要依次进入串口中断分别读取 4 位数字，接收到 4 位数字后置相应标志位，此时主程序执行判断数据有效、刷新显示数值、发送门票价格操作，其中发送门票价格同样需要依次发送各位数字。

系统主程序需要实现液晶显示和发送数据相关的控制功能，程序设计流程图如图 7.6(a)所示，程序设计时需要注意等待发送结束并手动清零置 TI=0。串口通信需要实现与接收数据相关的控制功能，程序设计流程图如图 7.6(b)所示，其中 k1 为接收次数。设定 bit 类型的座位标志位 seat_flag，当串口中断接收 4 次时置 seat_flag=1，此时主程序判断数据有效并置 seat_flag=0，如果数据有效则执行后续操作，执行结束置 status_flag=2。

针对本步骤的设计任务，设计思路应首先考虑实现串口通信部分，接收 4 位数字并置座位标志位，然后在此基础上实现系统主程序部分，显示座位选择界面，根据座位标志位判断数据有效，执行后续操作并切换系统状态。

（1）定义 seat_flag 为座位标志位，定义 recv[4]为接收数组，定义 k1 为接收次数，根据座位选择模块程序设计流程图完善串口通信函数。相应程序如下：

```
...
bit seat_flag;                          //座位标志位定义
uchar recv[4];                          //接收数组定义

void uart() interrupt 4                 //串口通信函数
{
    uchar k1;                           //接收次数定义
    if(RI==1)                           //接收标志位为 1
    {
        RI=0;                           //接收标志位清零
        recv[k1]=SBUF;                  //保存接收数字
        k1++;                           //接收次数计数
        if(k1==4)                       //接收次数为 4
        {
            k1=0;                       //接收次数清零
            seat_flag=1;                //置座位标志位为 1
        }
    }
}
```

（2）在此基础上继续完成系统主程序部分。定义所需的显示内容为 code 型字符串数组，定义 row、num、price 分别为座位排数、座位号码、门票价格。指定第 1 行第 1 列显示赛事名称 sport，第 2 行第 1 列显示座位信息 seat，当座位标志位为 1 时判断数据是否有效，然后根据功能要求分别计算座位排数、座位号码、门票价格，建立并调用座位显示函数和价格发送函数，最后置状态标志位为 2 进入“订单确认”模块。相应程序如下：

```
...
uchar code seat[]   ={"row00num00RMB000"};     //座位信息 seat 定义
uchar row;                                     //座位排数定义
uchar num;                                     //座位号码定义
uint price;                                    //门票价格定义
void seat_disp();                              //座位显示函数声明
void price_send();                             //价格发送函数声明

void main()                                    //主函数
{
    ...
    while(1)
    {
        if(status_flag==0)                     //状态标志位为 0
        {
        }
        else if(status_flag==1)                //状态标志位为 1
        {
            lcd_pos(1,1);                      //显示位置第 1 行第 1 列
            sport_disp();                      //显示赛事名称
            lcd_pos(2,1);                      //显示位置第 2 行第 1 列
            lcdwrite_string(seat);             //显示座位选择 seat
            if(seat_flag==1)                   //座位标志位为 1
            {
                seat_flag=0;                   //置座位标志位为 0
                if(recv[0]+recv[1]!=0&&recv[2]+recv[3]!=0)
                                                       //判断数据有效
                {
                    row=recv[0]*10+recv[1];            //计算座位排数
                    num=recv[2]*10+recv[3];            //计算座位号码
                    price=1000-5*row;                  //计算门票价格
                    seat_disp();                       //座位显示
                    price_send();                      //价格发送
                    status_flag=2;                     //置状态标志位为 2
                }
```

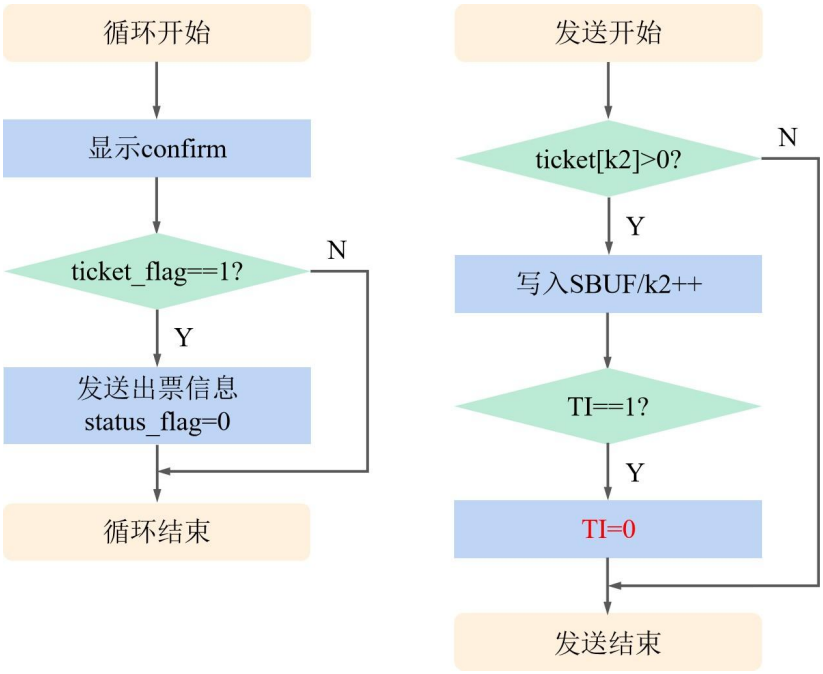
```
            }
        }
        else if(status_flag==2)          //状态标志位为 2
        {
        }
    }
}

void seat_disp()                         //座位显示函数
{
    lcd_pos(2,4);                        //显示位置第 2 行第 4 列
    lcdwrite_sz(row/10);
    lcdwrite_sz(row%10);                 //座位排数刷新显示
    lcd_pos(2,9);                        //显示位置第 2 行第 9 列
    lcdwrite_sz(num/10);
    lcdwrite_sz(num%10);                 //座位号码刷新显示
    lcd_pos(2,14);                       //显示位置第 2 行第 14 列
    lcdwrite_sz(price/100);
    lcdwrite_sz(price/10%10);
    lcdwrite_sz(price%10);               //门票价格刷新显示
}

void price_send()                        //价格发送函数
{
    SBUF=price/100+48;                   //发送价格百位
    while(TI==0);                        //等待发送结束
    TI=0;                                //发送标志位清零
    SBUF=price/10%10+48;                 //发送价格十位
    while(TI==0);                        //等待发送结束
    TI=0;                                //发送标志位清零
    SBUF=price%10+48;                    //发送价格个位
    while(TI==0);                        //等待发送结束
    TI=0;                                //发送标志位清零
}
```

4.2.4 step4：订单确认模块设计

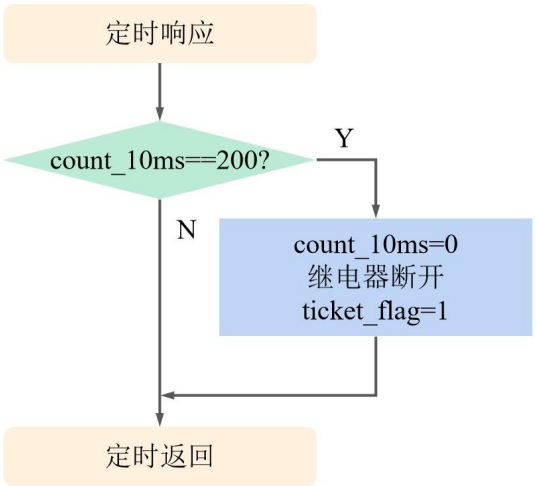
本步骤主要完成的具体任务是单片机显示订单确认界面，接收到确认信息后如果数字为 1 则继电器导通出票并返回出票信息，然后自动回到赛事选择界面，如果数字为 2 则直接回到赛事选择界面。其中与串口通信无关的控制功能为显示订单确认界面，与串口通信相关的控制功能为单片机接收 1 位数字和单片机发送

出票信息。

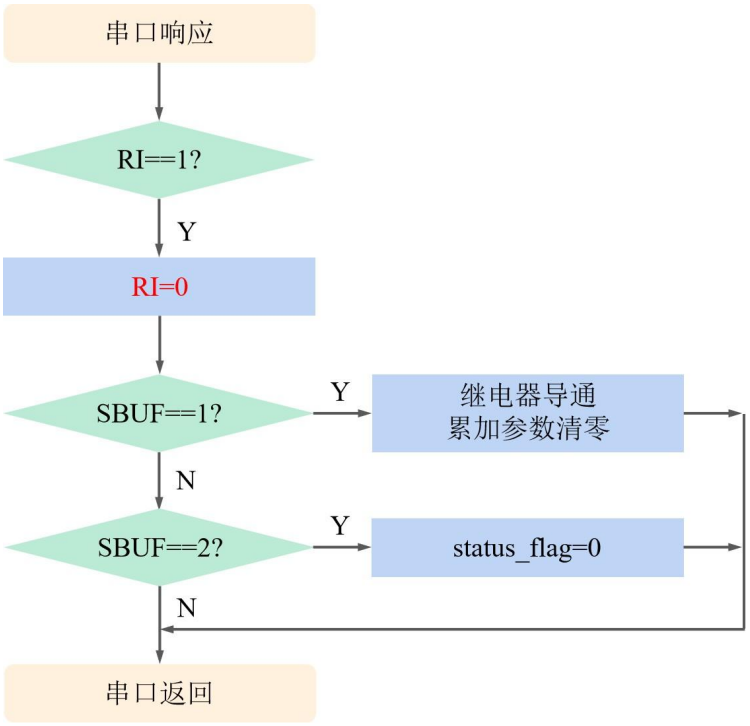
系统主程序需要实现液晶显示和发送数据相关的控制功能，程序设计流程图如图 7.7(a)所示。发送出票信息同样需要依次发送各位字符，其中 ticket[]为字符串数组，k2 为发送次数。定时器 T0 需要实现与定时相关的控制功能，程序设计流程图如图 7.7(b)所示。串口通信需要实现与接收数据相关的控制功能，程序设计流程图如图 7.7(c)所示。设定 bit 类型的出票标志位 ticket_flag，当定时中断定时 2s 时置 ticket_flag=1，此时主程序执行后续操作并置 ticket_flag=0，执行结束置 status_flag=0。



(a)



(b)



(c)

图 7.7 订单确认模块程序设计流程图

针对本步骤的设计任务，设计思路应首先考虑实现串口通信部分，继电器导通出票或置状态标志位，然后在此基础上实现定时器 T0 部分，定时参数累加然后继电器断开并置出票标志位，最后实现系统主程序部分，显示订单确认界面，根据出票标志位执行后续操作并切换系统状态。

（1）定义 issue 为继电器端口 P3.7，根据订单确认模块程序设计流程图完善串口通信函数。相应程序如下：

```
...
sbit issue=P3^7;                    //继电器端口 P3.7 定义

void uart() interrupt 4             //串口通信函数
{
    if(RI==1)                       //接收标志位为 1
    {
        RI=0;                       //接收标志位清零
        if(SBUF==1)                 //接收数字为 1
        {
            issue=0;                //继电器导通
            count=count_10ms=0;     //定时累加参数清零
        }
        else if(SBUF==2)            //接收数字为 2
```

```
            {
                status_flag=0;              //置状态标志位为 0
            }
        }
    }
```

（2）定义 ticket_flag 为出票标志位，根据订单确认模块程序设计流程图完善定时器 T0 函数。相应程序如下：

```
...
bit ticket_flag;                            //出票标志位定义

void timer0() interrupt 1                   //定时器 T0 函数
{
    count++;                                //定时累加
    if(count==100)                          //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==200)                 //定时 2s
        {
            count_10ms=0;
            if(issue==0)                    //继电器导通
            {
                issue=1;                    //继电器断开
                ticket_flag=1;              //置出票标志位为 1
            }
        }
    }
}
```

（3）最后完成系统主程序部分。定义所需的显示内容为 code 型字符串数组，指定第 1 行第 1 列显示确认信息 confirm，当出票标志位为 1 时建立并调用出票发送函数，最后置状态标志位为 0 回到“赛事选择”模块。相应程序如下：

```
...
uchar code confirm[]={"Confirm:1yes 2no"};      //确认信息 confirm 定义
uchar code ticket[]={" Ticket Issued! "};       //出票信息 ticket 定义
void ticket_send();                         //出票发送函数声明

void main()                                 //主函数
{
    ...
    while(1)
    {
        if(status_flag==0)                  //状态标志位为 0
```

```
        {
        }
        else if(status_flag==1)          //状态标志位为 1
        {
        }
        else if(status_flag==2)          //状态标志位为 2
        {
            lcd_pos(1,1);                //显示位置第 1 行第 1 列
            lcdwrite_string(confirm);    //显示确认信息 confirm
            if(ticket_flag==1)           //出票标志位为 1
            {
                ticket_flag=0;           //置出票标志位为 0
                ticket_send();           //出票发送
                status_flag=0;           //置状态标志位为 0
            }
        }
    }
}

void ticket_send()                       //出票发送函数
{
    uchar k2;                            //发送次数定义
    for(k2=0;ticket[k2]>0;k2++)
    {
        SBUF=ticket[k2];                 //发送字符串 k2 位
        while(TI==0);                    //等待发送结束
        TI=0;                            //发送标志位清零
    }
}
```

4.3 系统软件程序

综上所述，本例供参考的软件程序如下：

```
#include<reg51.h>
#include<intrins.h>
#include"1602LCD.h"
#define uchar unsigned char
#define uint unsigned int
uchar code name[] ={"19th Asian Games"};          //系统名称 name 定义
uchar code disp[]  ={" U ARE WELLCOME "};         //欢迎标语 disp 定义
uchar code list[]   ={"Competition List"};        //界面名称 list 定义
uchar code sport1[]={"   1.Athletics     "};      //赛事名称 sport1 定义
uchar code sport2[]={"   2.Gymnastics   "};       //赛事名称 sport2 定义
```

```
uchar code sport3[]={"    3.Basketball    "};          //赛事名称 sport3 定义
uchar code sport4[]={"    4.Swimming       "};         //赛事名称 sport4 定义
uchar code seat[]   ={"row00num00RMB000"};     //座位信息 seat 定义
uchar code confirm[]={"Confirm:1yes 2no"};        //确认信息 confirm 定义
uchar code ticket[]={" Ticket Issued! "};             //出票信息 ticket 定义
sbit issue=P3^7;                          //继电器端口 P3.7 定义
uchar status_flag;                        //状态标志位定义
uchar sport_flag;                         //赛事标志位定义
bit seat_flag;                            //座位标志位定义
bit ticket_flag;                          //出票标志位定义
uchar recv[4];                            //接收数组定义
uchar row;                                //座位排数定义
uchar seat;                               //座位号码定义
uchar price;                              //门票价格定义
uchar count,count_10ms;                   //定时累加参数定义
void sport_disp();                        //赛事显示函数声明
void seat_disp();                         //座位显示函数声明
void price_send();                        //价格发送函数声明
void ticket_send();                       //出票发送函数声明
void Delay1000ms();                       //延时 1000ms 函数声明

void main()
{
    lcd_init();                           //液晶显示初始化
    lcd_pos(1,1);                         //显示位置第 1 行第 1 列
    lcdwrite_string(name);                //显示系统名称 name
    lcd_pos(2,1);                         //显示位置第 2 行第 1 列
    lcdwrite_string(disp);                //显示欢迎标语 disp
    Delay1000ms();                        //延时 1000ms
    lcd_clear();                          //液晶显示清屏
    EA=1;                                 //打开总中断允许开关
    ES=1;                                 //打开串口通信开关
    ET0=1;                                //打开定时器 T0 开关
    TMOD=0x52;                            //设置定时器 T0 为方式 2 定时
                                            设置定时器 T1 为方式 2 定时
    TL0=0xA4;
    TH0=0xA4;                             //设置定时器 T0 初值
    TR0=1;                                //启动定时器 T0
    SCON=0x50;                            //设置串口通信为方式 1 且允许接收
    TL1=0xFD;
    TH1=0xFD;                             //设置定时器 T1 初值
    TR1=1;                                //启动定时器 T1
    while(1)
    {
```

```
        if(status_flag==0)              //状态标志位为 0
        {
            lcd_pos(1,1);               //显示位置第 1 行第 1 列
            lcdwrite_string(list);      //显示界面名称 list
            lcd_pos(2,1);               //显示位置第 2 行第 1 列
            sport_disp();               //显示赛事名称
        }
        else if(status_flag==1)         //状态标志位为 1
        {
            lcd_pos(1,1);               //显示位置第 1 行第 1 列
            sport_disp();               //显示赛事名称
            lcd_pos(2,1);               //显示位置第 2 行第 1 列
            lcdwrite_string(seat);      //显示座位信息 seat
            if(seat_flag==1)            //座位标志位为 1
            {
                seat_flag=0;            //置座位标志位为 0
                if(recv[0]+recv[1]!=0&&recv[2]+recv[3]!=0)
                {
                    row=recv[0]*10+recv[1];       //计算座位排数
                    num=recv[2]*10+recv[3];       //计算座位号码
                    price=1000-5*row;             //计算门票价格
                    seat_disp();                  //座位显示
                    price_send();                 //价格发送
                    status_flag=2;                //置状态标志位为 2
                }
            }
        }
        else if(status_flag==2)         //状态标志位为 2
        {
            lcd_pos(1,1);               //显示位置第 1 行第 1 列
            lcdwrite_string(confirm);   //显示确认信息 confirm
            if(ticket_flag==1)          //出票标志位为 1
            {
                ticket_flag=0;          //置出票标志位为 0
                ticket_send();          //出票发送
                status_flag=0;          //置状态标志位为 0
            }
        }
    }
}

void sport_disp()                       //赛事显示函数
{
    switch(sport_flag)
```

```
    {
        case 0:lcdwrite_string(sport1);break;          //标志位为 0 显示 sport1
        case 1:lcdwrite_string(sport2);break;          //标志位为 1 显示 sport2
        case 2:lcdwrite_string(sport3);break;          //标志位为 2 显示 sport3
        case 3:lcdwrite_string(sport4);break;          //标志位为 3 显示 sport4
    }
}

void seat_disp()                        //座位显示函数
{
    lcd_pos(2,4);                       //显示位置第 2 行第 4 列
    lcdwrite_sz(row/10);
    lcdwrite_sz(row%10);                //座位排数刷新显示
    lcd_pos(2,9);                       //显示位置第 2 行第 9 列
    lcdwrite_sz(num/10);
    lcdwrite_sz(num%10);                //座位号码刷新显示
    lcd_pos(2,14);                      //显示位置第 2 行第 14 列
    lcdwrite_sz(price/100);
    lcdwrite_sz(price/10%10);
    lcdwrite_sz(price%10);              //门票价格刷新显示
}

void price_send()                       //价格发送函数
{
    SBUF=price/100+48;                  //发送价格百位
    while(TI==0);                       //等待发送结束
    TI=0;                               //发送标志位清零
    SBUF=price/10%10+48;                //发送价格十位
    while(TI==0);                       //等待发送结束
    TI=0;                               //发送标志位清零
    SBUF=price%10+48;                   //发送价格个位
    while(TI==0);                       //等待发送结束
    TI=0;                               //发送标志位清零
}

void ticket_send()                      //出票发送函数
{
    uchar k2;                           //发送次数定义
    for(k2=0;ticket[k2]>0;k2++)
    {
        SBUF=ticket[k2];                //发送字符串 k2 位
        while(TI==0);                   //等待发送结束
        TI=0;                           //发送标志位清零
    }
```

```
}

void timer0() interrupt 1                    //定时器 T0 函数
{
    count++;                                 //定时累加
    if(count==100)                           //定时 10ms
    {
        count=0;
        count_10ms++;
        if(count_10ms==200)                  //定时 2s
        {
            count_10ms=0;
            if(status_flag==0)               //状态标志位为 0
            {
                sport_flag++;                //运动标志位累加
                if(sport_flag==4)            //运动标志位为 4
                {
                    sport_flag=0;            //运动标志位清零
                }
            }
            else if(status_flag==2)          //状态标志位为 2
            {
                if(issue==0)                 //继电器导通
                {
                    issue=1;                 //继电器断开
                    ticket_flag=1;           //置出票标志位为 1
                }
            }
        }
    }
}

void uart() interrupt 4                      //串口通信函数
{
    uchar k1;                                //接收次数定义
    if(RI==1)                                //接收标志位为 1
    {
        RI=0;                                //接收标志位清零
        if(status_flag==0)                   //状态标志位为 0
        {
            switch(SBUF)
            {
                case 1:sport_flag=0;status_flag=1;break;
                case 2:sport_flag=1;status_flag=1;break;
```

```
                case 3:sport_flag=2;status_flag=1;break;
                case 4:sport_flag=3;status_flag=1;break;
            }//SBUF 为 0/1/2/3 置赛事标志位为 1/2/3/4 并置状态标志位为 1
        }
        else if(status_flag==1)          //状态标志位为 1
        {
            recv[k1]=SBUF;               //保存接收数字
            k1++;                        //接收次数计数
            if(k1==4)                    //接收次数为 4
            {
                k1=0;                    //接收次数清零
                seat_flag=1;             //置座位标志位为 1
            }
        }
        else if(status_flag==2)          //状态标志位为 2
        {
            if(SBUF==1)                  //接收数字为 1
            {
                issue=0;                 //继电器导通
                count=count_10ms=0;      //定时累加参数清零
            }
            else if(SBUF==2)             //接收数字为 2
            {
                status_flag=0;           //置状态标志位为 0
            }
        }
    }
}

void Delay1000ms()                       //延时 1000ms 函数
{
    unsigned char i, j, k;
    _nop_();
    i = 8;
    j = 1;
    k = 243;
    do
    {
        do
        {
            while (--k);
        } while (--j);
    } while (--i);
}
```

5. 本章小结

本章设计开发了一种亚运会赛事售票控制系统，结合前面章节所学知识使用串口通信实现用户计算机进行赛事选择、座位选择、订单确认和后台单片机进行自动响应的售票过程。使用串口通信存在一些注意事项和使用技巧，总结如下：（1）串口初始化配置包括设置寄存器 SCON 和打开中断允许开关，当波特率由定时器控制时还需要进行定时初始化配置，包括设置寄存器 TMOD、设置定时初值、启动定时器，根据功能要求灵活进行配置；（2）串口通信接收数据结束时会置接收标志位 RI=1，发送数据结束时会置发送标志位 TI=1，产生中断请求触发中断响应，同时必须由程序手动清零；（3）串口通信以字节为单位进行数据传输，即单次收发过程仅为 1 位数字、字母、符号，通过多次接收和发送数据实现多位数据传输。希望读者在学习本章知识和完成设计实践后能够在不同的实例中灵活应用串口通信的控制功能，并能够应用前面章节所学知识独立完成综合控制系统的设计工作。

另外，本章要求的控制功能还有待进一步丰富，程序设计还可以进一步优化，请感兴趣的读者继续思考和开展后续设计。

6.拓展训练

在本章已完成系统的基础上，应用前面章节所学知识继续完善亚运会控制系统的后续功能，例如场馆检票控制功能、网络安保控制功能、智能灯光控制功能等，具体功能自拟。