目录

[一. 时钟 2](#_Toc131071379)

[1.1 项目简介 2](#_Toc131071380)

[1.1.1 硬件简介 2](#_Toc131071381)

[1.1.2 使用说明 3](#_Toc131071382)

[1.2 设计思路 4](#_Toc131071383)

[1.2.1 硬件设计 4](#_Toc131071384)

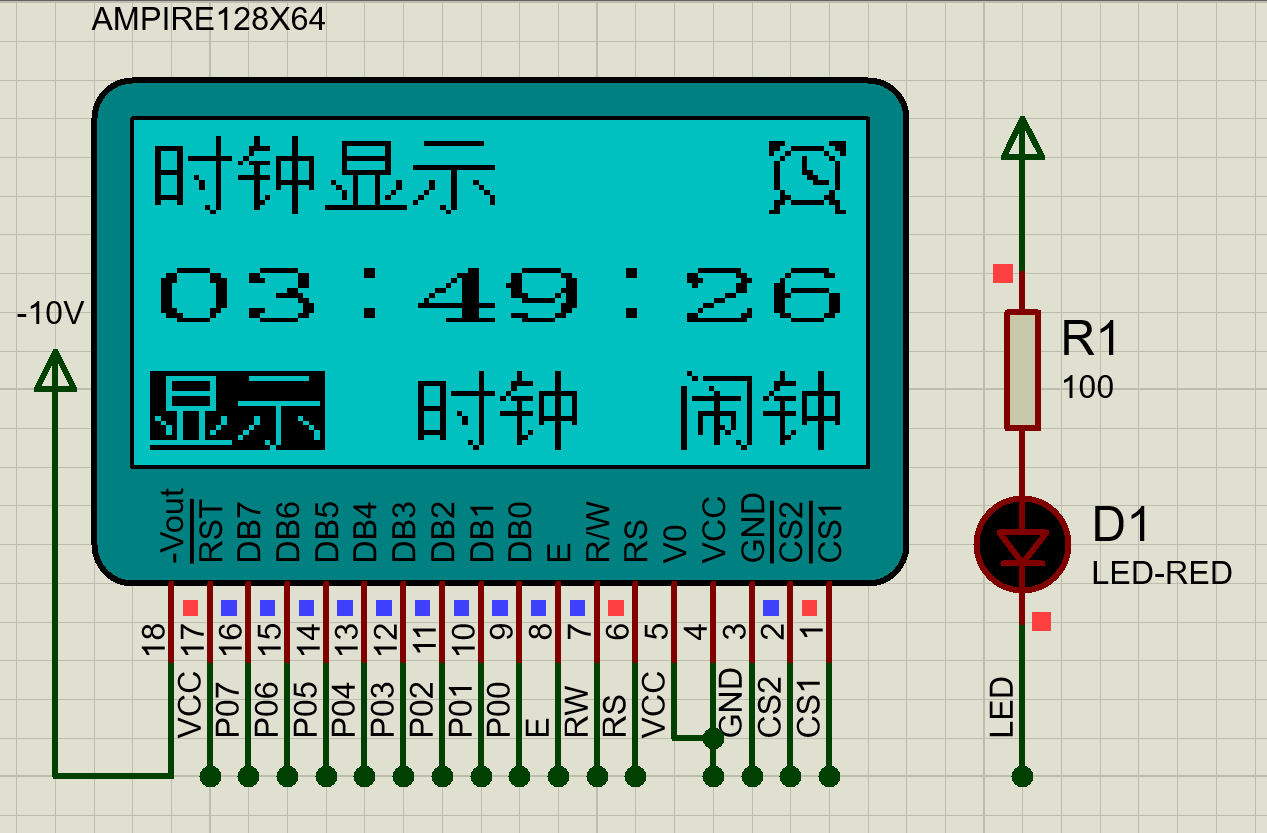
[1.2.2 软件设计 4](#_Toc131071385)

# 时钟

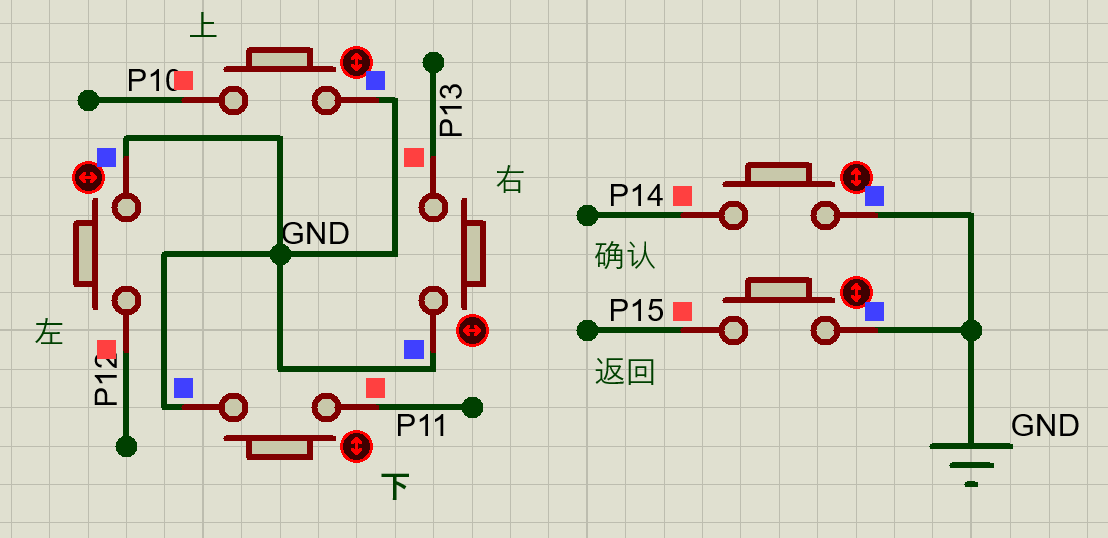
## 1.1 项目简介

### 1.1.1 硬件简介

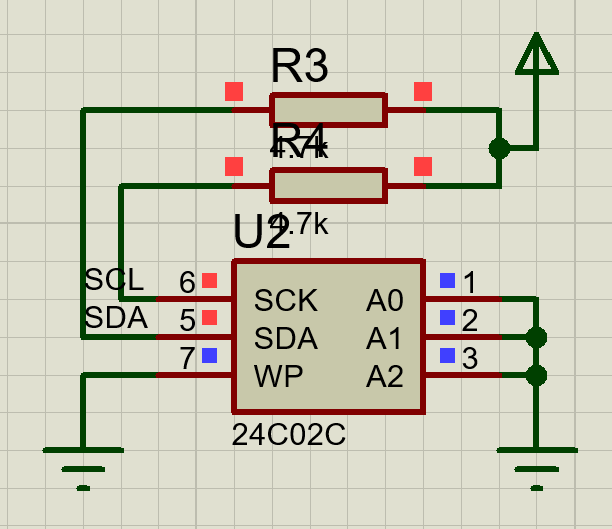
* LCD12864主显示界面 + LED闹钟提醒灯



* 键盘控制：四个方向键 + 两个功能键



* 外接EEPROM实现数据断电保存：包括时钟、闹钟



* 状态显示：时钟共有三个状态：时钟显示（正常显示时钟的运行）、时钟调整（修改时钟的时间）、闹钟调整（设置、取消闹钟）。这三种状态都会在屏幕的左上角进行显示提示用户当前处于哪一种状态
* 状态选择：通过按键控制屏幕最下方光标选择不同功能（显示、时钟、闹钟）对应三种状态，被光标选中的功能会反向显示
* 闹钟提醒：屏幕右上角的小闹钟图标实时标记闹钟是否开启，若闹钟到点，则LED灯将会已闪烁三次的方式提醒用户，并在闪烁完成后熄灭LED灯和关闭闹钟

### 1.1.2 使用说明

* 时钟显示：使用方向键控制光标位于“显示”处，按下确认键，时钟将退出其他状态，进行时钟的正常运行，显示最小以秒为单位的时间
* 时钟调整：

使用方向键控制光标位于“时间”处，按下确认键，时钟将进入修改当前时间的状态，用户可以通过方向键选择时、分、秒并修改其数值。

1. 若再次按下确认键则确认修改调整的时间为当前时钟时间，屏幕将显示“时间设置成功”
2. 若中途按下返回键则取消修改时间，屏幕将显示“退出时钟调整”，系统的时钟不会被修改

用户可通过移动光标选择“显示”查看时间的运行

* 闹钟调整：

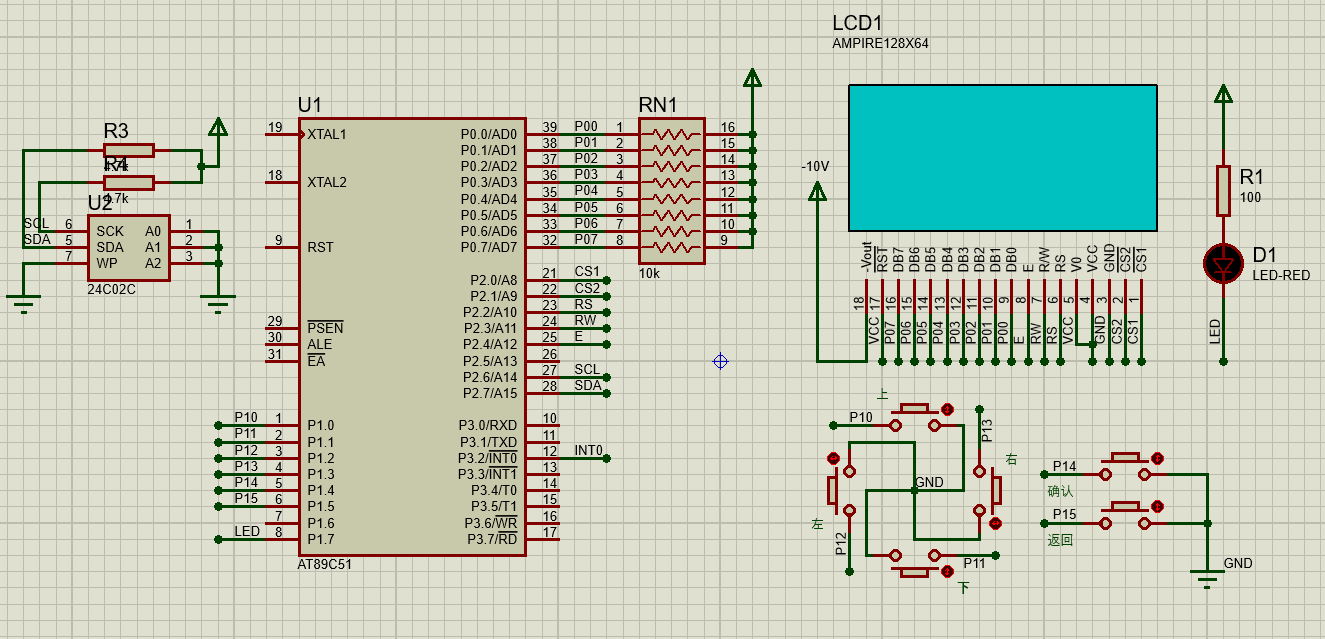
使用方向键控制光标位于“闹钟”处，按下确认键，时钟将进入设置闹钟的状态，用户可以通过方向键选择时、分、秒并修改闹钟的数值

1. 若再次按下确认键则确认设置闹钟，屏幕将显示“闹钟设置成功”，并在屏幕右上角浮现一个小闹钟的图案，标记闹钟已开启
2. 若中涂按下返回键则取消、关闭闹钟，屏幕将显示“退出闹钟调整”，并抹去屏幕右上角一个小闹钟的图标（如果一开始有的话），标记闹钟已关闭

用户可通过移动光标选择“显示”查看时间的运行，并通过查看屏幕右上角是否有一个小闹钟标识来确认闹钟是否有开启

## 1.2 设计思路

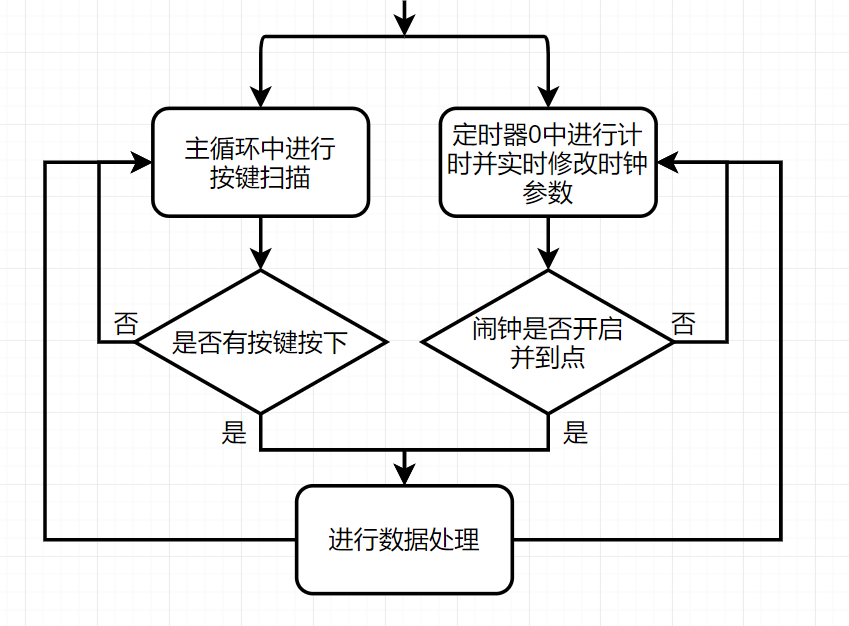
### 1.2.1 硬件设计



1. AT89C51单片机：该小项目需要用到的硬件资源并不多，只需要足够的IO口和一个定时器既可完成，选用最简单的51单片机既可满足
2. LCD12864：完成一个时钟需要进行最基础的六个数字的显示，还需要显示一些必要的文字给予用户更好的交互体验，选用数码管无法进行文字的显示，而更大屏幕的显示屏则没有必要
3. 24C02：选用最常见的EEPROM来储存时钟运行的基础数据，以实现断电数据保存

### 1.2.2 软件设计

* Main函数中：



* 定时器0中的计时：

1. 在程序开始运行时初始化T0开始工作在方式1：16位定时器
2. 在T0的中断服务函数中进行一秒钟的计时，并处理时分秒的进位计算，同时还判断闹钟功能是否开启并到点

//初始化定时器0

void InitTimer0()

{

TMOD = 0x01; //0000 0001 //T0工作在方式1：16位定时器

TH0 = (65536 - 50200) / 256; //50000us 256=2^8

TL0 = (65536 - 50200) % 256; //取低8位

ET0 = 1; //开启定时器0的中断

EA = 1; //开启总中断

TR0 = 1; //启动定时器0

}

//定时器0中断执行函数

void Timer0\_isr() interrupt 1

{

//进行初值重载

TH0 = (65536 - 50200) / 256;

TL0 = (65536 - 50200) % 256;

//进入中断

cnt++;

//只有进行了一秒的周期之后才会进行一次数据处理

if (cnt == 16) //决定了一秒的时间长度 //理论值为50000:20，根据实际情况调试

{

cnt = 0; //计数器归零

ModifyTimeInterrupt(); //修改时间

}

}

1. 对一秒钟的计算：理论值应该是50000us \* 20次，但是由于程序在T0的中断服务函数中要进行一定的计算，该部分会占用一定的运行时间，所以具体的值是通过多次不断调试得出的较为接近的结果：50200us \* 16次

* 按键扫描函数：

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: KeyScan

\* 功能说明: 获取用户按下的按键信号，并返回对应的按键标志

\* 形 参: 无

\* 返 回 值: 1. ENTER\_PRES : 确认键

\* 2. UP\_PRES : 向上键

\* 3. DOWN\_PRES : 向下键

\* 4. LEFT\_PRES : 向左键

\* 5. RIGHT\_PRES : 向右键

\* 6. RETURN\_PRES: 返回键

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int KeyScan()

{

unsigned char key\_flag = 1; //按键松开标志

//ModifyTime();

if (key\_flag == 1 && (key\_enter == 0 || key\_up == 0 || key\_down == 0 || key\_left == 0 || key\_right == 0 || key\_return == 0))

{

/\*

\* 把这个保存数据的功能放在这里而非主循环中是因为：该函数的缺点为运行时间较长，

\* 若放在主循环中会影响到单片机对按键的响应速度，导致有时候出现按键不响应的情况，

\* 放在这里既可以消除该影响，还能利用该函数的运行时间再配合延时函数做一个简单的按键消抖

\* （若不调用该函数，则延时应为delay(200)）

\*/

delay(130); //消抖

WriteTimeEEPROM(); //把数据写进EEPROM中

key\_flag = 0;

if (key\_enter == 0) return ENTER\_PRES; //确认键

else if (key\_return == 0) return RETURN\_PRES; //返回键

else if (key\_up == 0) return UP\_PRES; //向上键

else if (key\_down == 0) return DOWN\_PRES; //向下键

else if (key\_left == 0) return LEFT\_PRES; //向左键

else if (key\_right == 0) return RIGHT\_PRES; //向右键

}

else if (key\_enter == 0 && key\_up == 0 && key\_down == 0 && key\_left == 0 && key\_right == 0 && key\_return == 0)

{

key\_flag = 1;

}

/\*

\* 由于按键并非时刻有被按下，也意味着数据并非真正的时刻保存，

\* 因此除了充当消抖的延时函数外，再在无按键按下的情况进行调用，

\* 实测不会影响到按键的响应问题，而数据的保存也是实时的

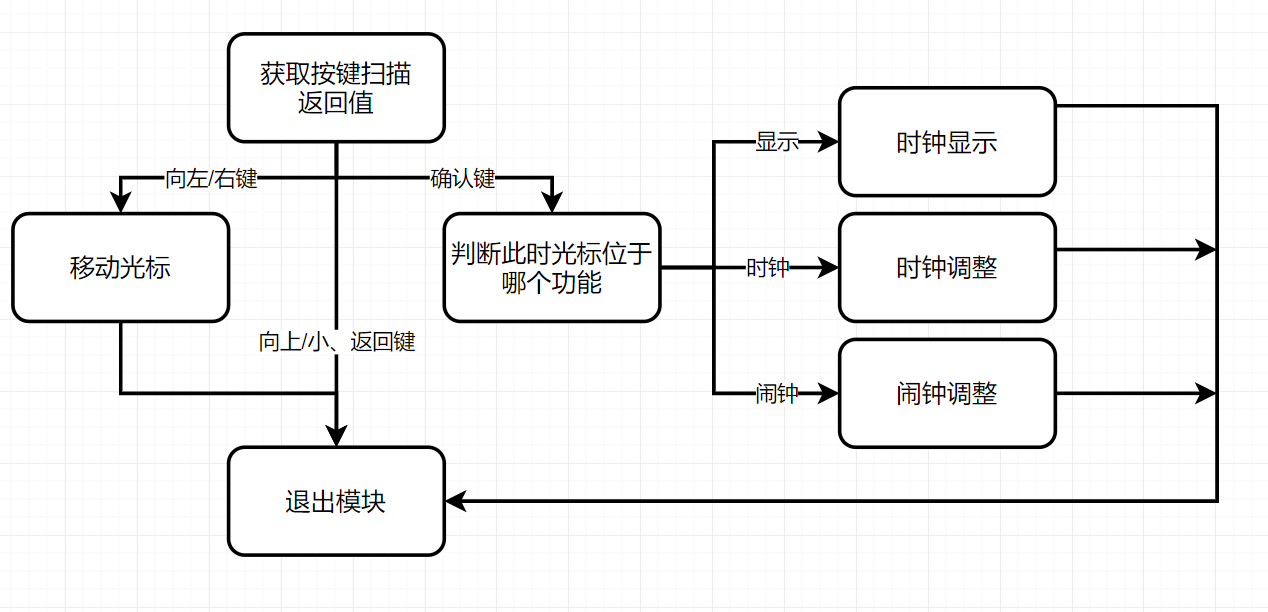
\*/

WriteTimeEEPROM(); //把数据写进EEPROM中

return 0; //无按键按下

}

* 主菜单



1. 主菜单中获取一次按键的扫描结果，并对该结果进行处理
2. 该函数在主循环中反复调用

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: Menu0

\* 功能说明: 通过获取用户按下的按键标志，实现对应的功能（确认键：进入对应功能，向左/右键：移动光标）

\* 形 参: 无

\* 返 回 值: 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void Menu0()

{

key = KeyScan(); //获取用户按下的按键

switch (key) //判断按下的是哪个按键

{

case ENTER\_PRES: { //确认键

switch (cursor) //判断按下确认键时光标在哪个功能

{

case 0: { //时钟显示

flag = 1;

//显示时间

PrintTimeHour(my\_time.hour);

PrintTimeMinute(my\_time.minute);

PrintTimeSeconds(my\_time.seconds);

//有时候LCD屏出现花屏，可以通过不断点击该功能进行刷新

PrintColon();

PrintCursor(cursor); //在LCD上显示光标

PrintClock(alarm\_time.flag); //在LCD上显示闹钟

PrintMode(cursor); //在LCD上显示时钟显示

break;

}

case 1: { //时钟调整

flag = 0;

PrintMode(cursor); //在LCD上显示时钟调整

PrintColon();

SetClock(); //调整时钟

break;

}

case 2: { //闹钟模调整

flag = 0;

PrintMode(cursor); //在LCD上显示闹钟调整

PrintColon();

SetAlarmClock(); //调整闹钟

break;

}

default:break;

}

break;

}

case LEFT\_PRES: { //向左

if (cursor > 0)

{

cursor--;

PrintCursor(cursor);

}

break;

}

case RIGHT\_PRES: { //向右

if (cursor < 2)

{

cursor++;

PrintCursor(cursor);

}

break;

}

default:break; //向上、下，返回不给予反应

}

}

* 时钟、闹钟调整

1. 这两个功能的实现过程相近，都是先用一个临时变量来储存用户修改后的时间参数，等待用户按下确认键后再将该临时变量的值赋值给真正的时钟/闹钟，若用户按下返回键则不将该值赋值，取消时钟/闹钟的调整过程
2. 这两个功能中对选中参数的光标依然使用反向显示的方式，多出使用到switch的嵌套

//设置闹钟

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: SetAlarmClock

\* 功能说明: 不断循环获取用户按下的按键标志（按下确认键退出循环），修改闹钟的数值，被选中的参数会反向显示

\* 以提示用户

\* 形 参: 无

\* 返 回 值: 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void SetAlarmClock()

{

//获取旧闹钟的值并在LCD上显示

int hour = alarm\_time.hour;

int minute = alarm\_time.minute;

int seconds = alarm\_time.seconds;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

//进行按键扫描，获取闹钟值

while (1)

{

key = KeyScan(); //获取用户按下的按键

switch (key)

{

case ENTER\_PRES: { //确认

PrintSetOK(1); //在LCD上显示闹钟设置成功

//退出按键扫描后，为闹钟赋值

alarm\_time.hour = hour;

alarm\_time.minute = minute;

alarm\_time.seconds = seconds;

alarm\_time.flag = 1; //开启闹钟

PrintClock(alarm\_time.flag);

return;

}

case RETURN\_PRES: { //返回

PrintReturn(1); //在LCD上显示退出闹钟调整

alarm\_time.flag = 0; //关闭闹钟

PrintClock(alarm\_time.flag);

return;

}

case UP\_PRES: { //向上

//调整参数：+1

switch (cursor\_num)

{

case 1: {

hour++;

if (hour == 24)

hour = 0;

PrintTimeHourReverse(hour);

break;

}

case 2: {

minute++;

if (minute == 60)

minute = 0;

PrintTimeMinuteReverse(minute);

break;

}

case 3: {

seconds++;

if (seconds == 60)

seconds = 0;

PrintTimeSecondsReverse(seconds);

break;

}

default:break;

}

break;

}

case DOWN\_PRES: { //向下

//调整参数：-1

switch (cursor\_num)

{

case 1: {

hour--;

if (hour == -1)

hour = 23;

PrintTimeHourReverse(hour);

break;

}

case 2: {

minute--;

if (minute == -1)

minute = 59;

PrintTimeMinuteReverse(minute);

break;

}

case 3: {

seconds--;

if (seconds == -1)

seconds = 59;

PrintTimeSecondsReverse(seconds);

break;

}

default:break;

}

break;

}

case LEFT\_PRES: { //向左

if (cursor\_num > 1)

cursor\_num--;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

break;

}

case RIGHT\_PRES: { //向右

if (cursor\_num < 3)

cursor\_num++;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

break;

}

default:break;

}

}

}

//设置时钟

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: SetClock

\* 功能说明: 不断循环获取用户按下的按键标志（按下确认键退出循环），修改时钟的数值，被选中的参数会反向显示

\* 以提示用户

\* 形 参: 无

\* 返 回 值: 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void SetClock()

{

//获取旧时钟的值并在LCD上显示

int hour = my\_time.hour;

int minute = my\_time.minute;

int seconds = my\_time.seconds;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

//进行按键扫描，获取闹钟值

while (1)

{

key = KeyScan(); //获取用户按下的按键

switch (key)

{

case ENTER\_PRES: { //确认

PrintSetOK(0); //在LCD上显示时钟设置成功

//退出按键扫描后，为时钟赋值

my\_time.hour = hour;

my\_time.minute = minute;

my\_time.seconds = seconds;

cnt = 0; //把定时器的计数器置0

return;

}

case RETURN\_PRES: { //返回

PrintReturn(0); //在LCD上显示退出时钟调整

return;

}

case UP\_PRES: { //向上

//调整参数：+1

switch (cursor\_num)

{

case 1: {

hour++;

if (hour == 24)

hour = 0;

PrintTimeHourReverse(hour);

break;

}

case 2: {

minute++;

if (minute == 60)

minute = 0;

PrintTimeMinuteReverse(minute);

break;

}

case 3: {

seconds++;

if (seconds == 60)

seconds = 0;

PrintTimeSecondsReverse(seconds);

break;

}

default:break;

}

break;

}

case DOWN\_PRES: { //向下

//调整参数：-1

switch (cursor\_num)

{

case 1: {

hour--;

if (hour == -1)

hour = 23;

PrintTimeHourReverse(hour);

break;

}

case 2: {

minute--;

if (minute == -1)

minute = 59;

PrintTimeMinuteReverse(minute);

break;

}

case 3: {

seconds--;

if (seconds == -1)

seconds = 59;

PrintTimeSecondsReverse(seconds);

break;

}

default:break;

}

break;

}

case LEFT\_PRES: { //向左

if (cursor\_num > 1)

cursor\_num--;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

break;

}

case RIGHT\_PRES: { //向右

if (cursor\_num < 3)

cursor\_num++;

//反向显示选中的参数

switch (cursor\_num)

{

case 1: {

PrintTimeHourReverse(hour);

PrintTimeMinute(minute);

PrintTimeSeconds(seconds);

break;

}

case 2: {

PrintTimeHour(hour);

PrintTimeMinuteReverse(minute);

PrintTimeSeconds(seconds);

break;

}

case 3: {

PrintTimeHour(hour);

PrintTimeMinute(minute);

PrintTimeSecondsReverse(seconds);

break;

}

}

break;

}

default:break;

}

}

}

* 数据保存到EEPROM

1. 使用IIC通信协议读写EEPROM，由于51没有硬件IIC，使用软件模拟
2. 对数据的保存使用最简单粗暴的方法：把时钟、闹钟共七个数据保存到24C02的七个固定地址中。
3. 在程序的开始，从24C02的地址中依次读取数据到程序的时钟、闹钟中

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: ReadTimeEEPROM

\* 功能说明: 把时钟、闹钟的数据从EEPROM中读取出来，直接把数据写进全局变量my\_time,alarm\_time中

\* 形 参: 无

\* 返 回 值: 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void ReadTimeEEPROM()

{

IIC\_Init(); //IIC初始化

//写EEPROM地址

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa0); //写地址 //0xa0：1010 0000

IIC\_ACK(); //应答

//读第一个字节

IIC\_Write\_Byte(0x00); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读 //0xa1：1010 0001

IIC\_ACK(); //应答

my\_time.hour = IIC\_Read\_Byte(); //读数据

//读第二个字节

IIC\_Write\_Byte(0x01); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读

IIC\_ACK(); //应答

my\_time.minute = IIC\_Read\_Byte(); //读数据

//读第三个字节

IIC\_Write\_Byte(0x02); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读

IIC\_ACK(); //应答

my\_time.seconds = IIC\_Read\_Byte(); //读数据

//读第四个字节

IIC\_Write\_Byte(0x03); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读 //0xa1：1010 0001

IIC\_ACK(); //应答

alarm\_time.hour = IIC\_Read\_Byte(); //读数据

//读第五个字节

IIC\_Write\_Byte(0x04); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读

IIC\_ACK(); //应答

alarm\_time.minute = IIC\_Read\_Byte(); //读数据

//读第六个字节

IIC\_Write\_Byte(0x05); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读

IIC\_ACK(); //应答

alarm\_time.seconds = IIC\_Read\_Byte(); //读数据

//读第七个字节

IIC\_Write\_Byte(0x06); //数据地址

IIC\_ACK(); //应答

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa1); //7位地址位，加上最低位读写位，允许读

IIC\_ACK(); //应答

alarm\_time.flag = IIC\_Read\_Byte(); //读数据

IIC\_Stop(); //IIC结束信号

}

1. 每次保存数据时把这七个参数以覆盖的方式写入24C02固定的七个地址中

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* 函 数 名: WriteTimeEEPROM

\* 功能说明: 把时钟、闹钟的数据写进EEPROM中，直接从全局变量my\_time,alarm\_time中获取数据并写入

\* 形 参: 无

\* 返 回 值: 无

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void WriteTimeEEPROM()

{

IIC\_Init(); //IIC初始化

//写EEPROM地址

IIC\_Start(); //IIC起始信号

IIC\_Write\_Byte(0xa0); //写地址 //0xa0：1010 0000

IIC\_ACK(); //应答

//写第一个字节

IIC\_Write\_Byte(0x00); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(my\_time.hour); //写数据

IIC\_ACK(); //应答

//写第二个字节

IIC\_Write\_Byte(0x01); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(my\_time.minute); //写数据

IIC\_ACK(); //应答

//写第三个字节

IIC\_Write\_Byte(0x02); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(my\_time.seconds);//写数据

IIC\_ACK(); //应答

//写第四个字节

IIC\_Write\_Byte(0x03); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(alarm\_time.hour); //写数据

IIC\_ACK(); //应答

//写第五个字节

IIC\_Write\_Byte(0x04); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(alarm\_time.minute); //写数据

IIC\_ACK(); //应答

//写第六个字节

IIC\_Write\_Byte(0x05); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(alarm\_time.seconds); //写数据

IIC\_ACK(); //应答

//写第七个字节

IIC\_Write\_Byte(0x06); //写字节地址

IIC\_ACK(); //应答

IIC\_Write\_Byte(alarm\_time.flag); //写数据

IIC\_ACK(); //应答

IIC\_Stop(); //结束信号

}

1. 把保存数据的函数放到按键扫描函数中（见KeyScan函数））：为了实现数据的实时保存，必须要让该函数在每次参数发生变化之后便被调用到，即至少要每秒钟执行一次。若放到主循环中又会对按键的扫描产生影响（该函数的运行需要一定的时间，会发生有时候按键不响应的Bug），因此直接把该函数的调用放在按键扫描函数中（按键的扫描也要求是实时进行的），并将该函数充当一个延时函数，配合延时函数用于按键的消抖