

# 单片机原理及应用

## 课程设计报告书

课 题： LED 数码管显示的电子钟设计

姓 名： XXX

学 号： XXX

专 业： 电子信息工程

学 院： 信息工程学院

指导教师： XXX

完成日期： 2026 年 1 月

2026 年 1 月

## 目录

<b>1 引言</b>	<b>1</b>
1.1 课题背景 . . . . .	1
1.2 国内外发展现状 . . . . .	1
1.2.1 国外技术发展现状 . . . . .	1
1.2.2 国内技术发展现状 . . . . .	2
1.3 课题研究意义 . . . . .	2
1.4 设计目标与技术指标 . . . . .	2
<b>2 系统总体方案设计</b>	<b>3</b>
2.1 系统功能需求分析 . . . . .	3
2.2 方案论证与比较 . . . . .	3
2.2.1 主控制器选型 . . . . .	3
2.2.2 计时方案选型 . . . . .	4
2.2.3 显示器件选型 . . . . .	4
2.3 系统总体架构设计 . . . . .	5
<b>3 硬件电路详细设计</b>	<b>5</b>
3.1 AT89C51 单片机核心资源详解 . . . . .	5
3.1.1 存储器组织结构 . . . . .	5
3.1.2 中断系统 . . . . .	6
3.2 DS1302 时钟电路深度解析 . . . . .	7
3.2.1 DS1302 内部寄存器结构 . . . . .	7
3.2.2 控制命令字 . . . . .	7
3.3 显示驱动电路设计 . . . . .	8
3.3.1 数码管静态与动态显示对比 . . . . .	8
3.3.2 驱动电流计算 . . . . .	8
3.4 按键输入电路 . . . . .	9
3.5 报警驱动电路原理 . . . . .	9
<b>4 软件系统详细设计</b>	<b>10</b>
4.1 软件开发环境与工具 . . . . .	10
4.2 程序总体架构 . . . . .	11
4.3 DS1302 通信时序的软件实现 . . . . .	12
4.4 按键消抖算法 . . . . .	12
4.5 数码管动态扫描中断服务程序 . . . . .	12

<b>5 系统仿真与调试</b>	<b>13</b>
5.1 Proteus 仿真环境搭建 . . . . .	13
5.2 调试过程与问题分析 . . . . .	13
5.2.1 问题一：数码管显示亮度不均或闪烁 . . . . .	13
5.2.2 问题二：DS1302 读出数据始终为 85 或 FF . . . . .	13
5.2.3 问题三：按键设置时数字跳变过快 . . . . .	13
5.2.4 问题四：继电器吸合后单片机复位 . . . . .	14
5.3 仿真结果 . . . . .	14
<b>6 使用说明书</b>	<b>15</b>
6.1 系统上电 . . . . .	15
6.2 功能键分布 . . . . .	15
6.3 操作流程示例 . . . . .	15
6.3.1 场景一：校对时间 . . . . .	15
6.3.2 场景二：设置闹钟 . . . . .	15
<b>7 总结与展望</b>	<b>16</b>
7.1 设计总结 . . . . .	16
7.2 存在的不足 . . . . .	16
7.3 未来展望 . . . . .	17
<b>8 参考文献</b>	<b>17</b>
<b>A 附录 A：系统源程序 (main.c)</b>	<b>18</b>
<b>B 附录 B：系统电路原理总图</b>	<b>27</b>

## 一、引言

### 1.1 课题背景

时间，作为物质运动的存在形式，是人类社会生活中最基本的物理量之一，也是科学的研究、工业生产、日常生活等各个领域不可或缺的重要参数。从古代的日晷、水钟、沙漏，到近代的机械钟，再到现代的石英钟、原子钟，人类对时间计量的精度和便捷性追求从未停止。每一次计时工具的革新，都代表着当时科技生产力的最高水平。

随着微电子技术的飞速发展，单片机（Microcontroller Unit, MCU）技术日益成熟，以其体积小、功耗低、功能强、性价比高等优势，在智能仪表、工业控制、家用电器、汽车电子等领域得到了广泛应用。基于单片机的电子钟不仅能实现基本的年月日时分秒计时功能，还能通过软件编程灵活地扩展功能，如多组闹钟、万年历、温湿度显示、农历转换、自动校时等，这是传统机械钟表无法比拟的。

在现代智能化社会，电子钟已不仅仅是一个简单的计时工具，更是一个集成了多种功能的智能终端节点。例如，在智能家居系统中，电子钟可以作为主控中心的时间基准，协调窗帘的定时开关、灯光的定时调节、空调的预约启动等；在工业现场，高精度的电子钟用于记录生产数据的时间戳，保证生产过程的可追溯性和同步性。因此，研究和设计基于单片机的电子钟系统，不仅具有重要的教学意义，能够帮助学生将理论知识转化为工程实践能力，也具有广泛的实际应用价值。

### 1.2 国内外发展现状

#### 1.2.1 国外技术发展现状

在国外，数字时钟技术已经非常成熟，并向着高精度、网络化和低功耗方向发展。

1. **高精度化：**高端电子钟广泛利用 GPS（全球定位系统）、GLONASS 或 Galileo 卫星信号进行自动授时，结合高稳定度的温补晶振（TCXO）甚至恒温晶振（OCXO），使得电子钟的时间误差可以控制在微秒级甚至纳秒级，广泛应用于金融交易、通信基站等对时间要求极高的领域。
2. **网络化与物联网：**随着 IoT 技术的普及，国外的智能时钟普遍具备 Wi-Fi、蓝牙或 ZigBee 通信能力，可以自动连接 NTP（网络时间协议）服务器进行校准，实现“永不准时”。同时，它们可以作为智能家居的控制面板，显示天气、股票信息，甚至集成语音助手（如 Alexa、Google Assistant）。
3. **极低功耗设计：**为了适应便携式和环保需求，采用了先进的低功耗微控制器和能量采集技术（如光能、动能），使得某些电子钟仅靠微弱的光线或温差即可长期运行，无需频繁更换电池。

### 1.2.2 国内技术发展现状

国内在电子钟领域的研究起步较晚，但发展迅速，产业链已经非常完善。特别是在单片机应用方面，国内高校和企业积累了丰富的经验。目前的市场产品主要分为两类：

1. **消费级市场：**以珠三角地区为代表，生产了全球绝大多数的数字电子钟。这类产品以低成本、功能多样化为主，广泛采用了国产的低成本单片机（如 STC、合泰等）和专用的时钟驱动芯片。虽然在核心算法和芯片制造工艺上与国外顶尖水平仍有差距，但在应用创新和外观设计上已经非常出色。
2. **专业级市场：**主要面向电力系统、轨道交通、医院等专用领域。国内企业已经研发出了基于北斗卫星授时的高精度子母钟系统，打破了国外的垄断，实现了关键基础设施的时间同步自主可控。

然而，我们在高端实时时钟芯片（RTC）的自主研发率，以及超低功耗模拟电路设计方面，仍有一定的提升空间，需要持续的技术积累。

### 1.3 课题研究意义

本次课程设计选取“LED 数码管显示的电子钟设计”作为课题，不仅仅是为了制作一个简单的计时器，更重要的是通过这个项目，达成以下教学与实践目标：

1. **巩固与深化理论知识：**将《单片机原理及应用》、《数字电子技术》、《模拟电子技术》、《C 语言程序设计》等课程的理论知识融会贯通。特别是加深对单片机内部哈佛结构、中断系统优先级、定时器/计数器工作模式、I/O 口扩展技术等核心概念的理解。
2. **提升工程实践能力：**通过原理图绘制、元器件选型与计算、电路焊接与调试、软件流程设计与编码等全过程的训练，锻炼学生的动手能力和解决实际工程问题的能力。这是从“纸上谈兵”到“实战演练”的关键转变。
3. **掌握通信协议与接口技术：**本设计涉及 DS1302 时钟芯片，通过对驱动程序的编写，可以深入理解 SPI (Serial Peripheral Interface) 串行通信协议的时序图、数据帧结构和编程方法，为后续学习 I2C、UART、USB、CAN 等复杂总线打下坚实基础。
4. **培养职业素养：**在设计过程中，需要查阅大量的英文数据手册 (Datasheet)，编写规范的技术文档，遵循代码编写规范，这些都是成为一名合格电子工程师所必须具备的职业素养。

### 1.4 设计目标与技术指标

本系统的设计目标是构建一个功能完善、运行稳定、操作简便、显示清晰的数字电子钟。具体技术指标如下：

- **主控芯片:** 采用经典的 MCS-51 内核单片机 AT89C51。
- **时钟源:** 采用专用实时时钟芯片 DS1302，外接 32.768kHz 晶振，具备闰年自动补偿功能。
- **显示方式:** 采用 8 位共阳极 LED 数码管，动态扫描显示，格式为“时时-分分-秒秒”(HH-MM-SS)。
- **输入方式:** 设计 4-6 个独立按键，实现时间修改、闹钟设置、模式切换等功能，要求按键响应灵敏，无误触发。
- **报警功能:** 具备闹钟功能，当设定时间到达时，驱动有源蜂鸣器发出声响，并控制继电器动作（模拟家电控制）。
- **掉电保护:** 系统需具备主备电源切换电路，当主电源 (+5V) 断电时，DS1302 由备用纽扣电池供电，保持内部时钟继续运行，上电后无需重新校时。
- **计时精度:** 日误差小于 1 秒。

## 二、系统总体方案设计

### 2.1 系统功能需求分析

根据任务书和实际使用场景，本电子钟系统需要满足以下功能需求：

1. **基本计时:** 能够准确记录并显示当前的小时、分钟和秒数。
2. **时间校准:** 由于初始上电或长期运行可能产生误差，必须提供人工校准功能。用户可以通过按键调整当前的小时和分钟。
3. **闹钟设置:** 用户可以设置一个或多个闹钟时间。
4. **闹钟响应:** 当实时时间与闹钟时间匹配时，系统应能产生声光报警信号，提醒用户。
5. **掉电走时:** 这是电子钟最关键的特性之一。传统单片机计时在断电后数据会丢失，因此必须引入外部 RTC 芯片及备用电源方案。
6. **人机交互:** 界面应直观，操作逻辑应简单易懂（例如设置时对应位闪烁）。

### 2.2 方案论证与比较

#### 2.2.1 主控制器选型

- **方案一:** 采用 STM32 系列单片机。STM32 基于 ARM Cortex-M 内核，主频高 (72MHz 甚至更高)，Flash 和 RAM 资源丰富，外设极其强大。但对于本电子钟项

目而言，其资源严重过剩，犹如“杀鸡用牛刀”。此外，STM32 芯片引脚多，焊接难度大，开发环境配置复杂，成本也相对较高。

- **方案二：采用 FPGA（现场可编程门阵列）。**利用 Verilog HDL 语言设计数字逻辑电路来实现计时。FPGA 并行处理能力强，扫描数码管非常稳定。但 FPGA 成本高昂，且断电后配置信息易丢失（需外挂 Flash），不适合做低成本电子钟。
- **方案三：采用 AT89C51/STC89C52 单片机。**这是经典的 8 位单片机，基于 MCS-51 指令集。具有 4KB/8KB Flash ROM, 128B/256B RAM, 32 个 I/O 口, 2-3 个定时器。其资源足以满足本设计的逻辑控制需求。更重要的是，51 单片机资料浩如烟海，价格低廉（几元人民币），封装友好（DIP40），非常适合教学入门和低成本制作。

**结论：**考虑到成本、开发难度和教学目的，选择方案三，使用 AT89C51 单片机作为主控核心。

### 2.2.2 计时方案选型

- **方案一：单片机内部定时器软件计时。**利用单片机内部的 Timer0 或 Timer1 产生固定周期的中断（如 50ms），通过软件计数累加实现时分秒。
  - 优点：硬件成本极低，无需额外芯片。
  - 缺点：占用 CPU 资源；晶振频率受温度影响大，精度难以保证；最致命的是，一旦单片机掉电或复位，时间将归零，必须重新设置，用户体验极差。
- **方案二：专用实时时钟芯片（RTC）。**采用 DS1302、DS1307 或 PCF8563 等专用芯片。
  - 优点：芯片内部集成了高精度的振荡器电路、分频电路和日历逻辑；具有独立的备用电源引脚，功耗极低（微安级），一颗纽扣电池可维持数年走时；单片机只需通过串行接口读取数据，无需干预计时过程。
  - 缺点：增加了一点硬件成本和 PCB 面积。

**结论：**为了保证计时精度和掉电走时这一核心功能，必须选择方案二。综合考虑性价比和接口复杂度（DS1302 为三线式，比 I2C 更简单直观），选用 DS1302 芯片。

### 2.2.3 显示器件选型

- **方案一：LCD1602 液晶显示屏。**可以显示两行字符（16x2），显示内容丰富，可显示英文提示，功耗极低。但 LCD1602 自身不发光，在夜间或暗处需要开启背光，且可视角度较小，机械强度较低，不耐摔。
- **方案二：OLED 显示屏。**自发光，对比度极高，显示效果华丽。但小尺寸 OLED 屏幕较贵，且长期显示固定内容容易产生“烧屏”现象，寿命不如 LED。

- **方案三：LED 数码管。**亮度高，红色光线醒目，可视角度极大，寿命长（可达 10 万小时），环境适应性强（耐高低温），结构坚固。这是市面上电子钟最主流的显示方案。

**结论：**本设计要求显示醒目，适合远距离观看，故选择方案三，使用 8 位共阳极 LED 数码管。

## 2.3 系统总体架构设计

基于上述分析，确定的系统采用模块化设计思想。系统核心由五大部分组成：

- **MCU 控制模块：**AT89C51，负责系统的整体调度。
- **时钟模块：**DS1302 + 32.768kHz 晶振 + 备用电池，负责底层计时。
- **显示模块：**74HC573（可选）+ 8 位共阳数码管，负责时间信息的呈现。
- **交互模块：**独立按键，负责用户指令的输入。
- **报警模块：**三极管 + 蜂鸣器 + 继电器，负责输出控制信号。

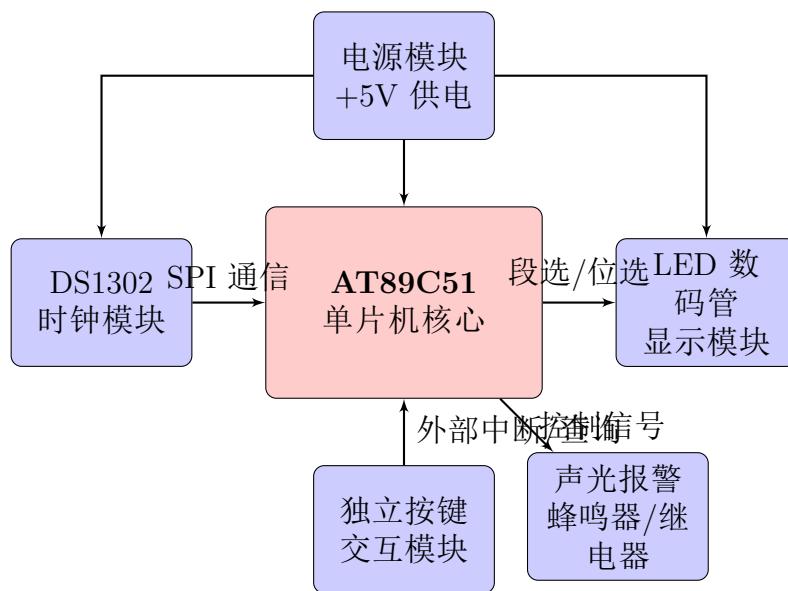


图 1：系统总体结构框图

## 三、硬件电路详细设计

### 3.1 AT89C51 单片机核心资源详解

AT89C51 是 Atmel 公司生产的低电压、高性能 CMOS 8 位微处理器，片内含 4KB 的可反复擦写的 Flash 只读程序存储器和 128B 的随机存取数据存储器（RAM）。

### 3.1.1 存储器组织结构

- 程序存储器 (ROM):** 地址范围 0000H-0FFFH。由于本系统代码量不大，4KB 空间绰绰有余。如果代码超过 4KB，可以通过 EA 引脚扩展外部 ROM。
- 数据存储器 (RAM):** 内部 128B RAM 分为三个区域：
  - 工作寄存器组 (00H-1FH):** 共 4 组，每组 8 个寄存器 (R0-R7)，通过 PSW 寄存器的 RS1、RS0 位切换。
  - 位寻址区 (20H-2FH):** 这 16 个字节的每一位都可以独立寻址，非常适合存放标志位（如闹钟开关标志、闪烁标志），能极大节省内存。
  - 通用 RAM 区 (30H-7FH):** 用于存放用户堆栈和暂存变量。

### 3.1.2 中断系统

AT89C51 提供 5 个中断源，分别是：1. 外部中断 0 ( $\overline{INT0}$ /P3.2) 2. 定时器 0 中断 (TF0) 3. 外部中断 1 ( $\overline{INT1}$ /P3.3) 4. 定时器 1 中断 (TF1) 5. 串口中断 (RI/TI)

在本设计中，我们使用了定时器 0 中断来实现数码管的动态扫描。因为数码管扫描需要极高的实时性（必须均匀地每隔几毫秒切换一位），如果放在主循环中，会被按键检测或 DS1302 通信的延时所阻塞，导致显示闪烁。使用中断可以保证显示的优先级最高，不受主程序逻辑影响。

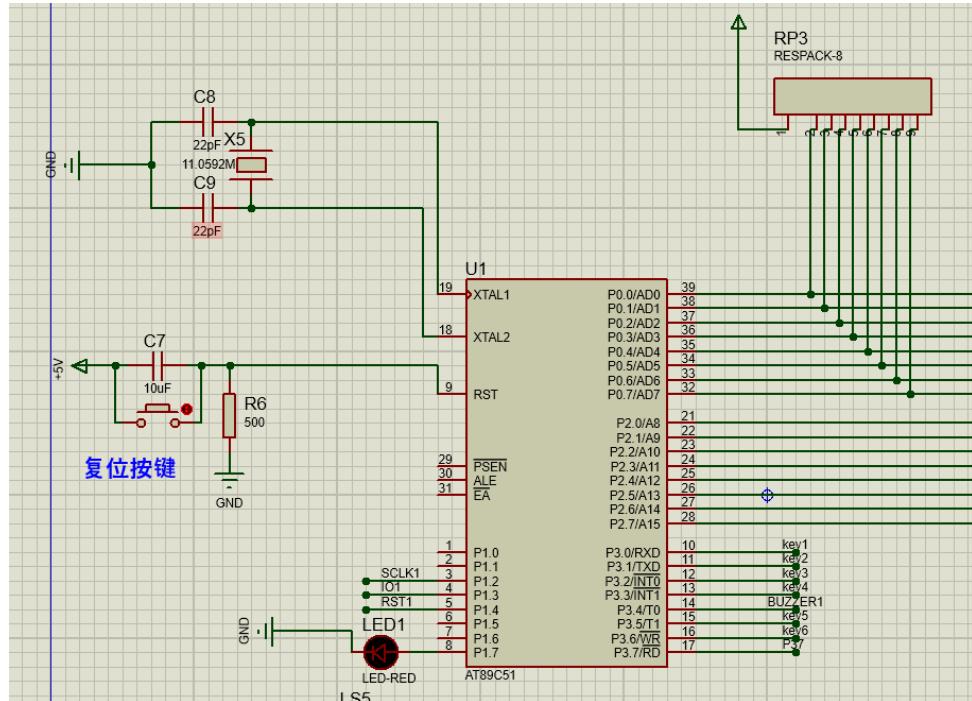


图 2: 单片机最小系统仿真图

如图 1 所示，单片机最小系统主要由电源电路、复位电路和时钟电路组成。

- 时钟电路：**连接在 XTAL1 (19 脚) 和 XTAL2 (18 脚) 之间的晶体振荡器 (Crystal Oscillator) 与两个负载电容 (C1, C2) 构成。本设计选用 12MHz 晶振，电容选用 30pF 瓷片电容。晶振产生的脉冲信号经过内部电路整形后，为 CPU 提供机器周期。一个机器周期等于 12 个时钟周期，即  $1\mu s$ ，这为定时器的计数计算提供了基准。
- 复位电路：**连接在 RST (9 脚) 引脚。采用上电复位电路，由  $10\mu F$  电解电容和  $10k\Omega$  电阻构成 RC 充电回路。上电瞬间，电容相当于短路，RST 脚获得高电平；随着电容充电，RST 脚电压逐渐下降至低电平，完成复位。

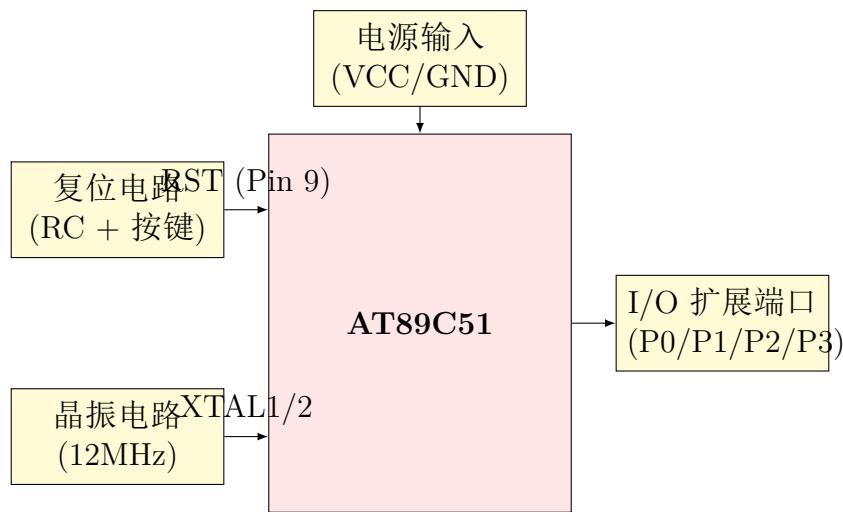


图 3: 单片机最小系统逻辑框图

## 3.2 DS1302 时钟电路深度解析

### 3.2.1 DS1302 内部寄存器结构

DS1302 拥有 31 字节的静态 RAM 和一组时钟/日历寄存器。时钟寄存器包含秒、分、时、日、月、周、年等信息。这些寄存器的数据格式为 **BCD 码** (Binary-Coded Decimal)。

- BCD 码特性：**用 4 位二进制数表示 1 位十进制数。例如，十进制数 59，在二进制中是 0011 1011，但在 BCD 码中，高 4 位表示十位 5 (0101)，低 4 位表示个位 9 (1001)，即 0101 1001 (0x59)。
- 读写转换：**单片机读取到 DS1302 的数据是 BCD 码，必须先转换为十进制才能进行加减运算；写入时，必须将十进制转换为 BCD 码。
  - BCD 转十进制： $dec = (bcd >> 4) \times 10 + (bcd \& 0x0F)$ ;
  - 十进制转 BCD： $bcd = (dec / 10) << 4 | (dec \% 10)$ ;

### 3.2.2 控制命令字

DS1302 的控制字为一个字节，格式如下：

- Bit 7: 必须为 1。
- Bit 6: 0 表示存取日历时钟数据，1 表示存取 RAM 数据。
- Bit 5-1: 寄存器地址。
- Bit 0: 0 表示写操作，1 表示读操作。

例如，秒寄存器的读地址为 0x81，写地址为 0x80；写保护寄存器的写地址为 0x8E。

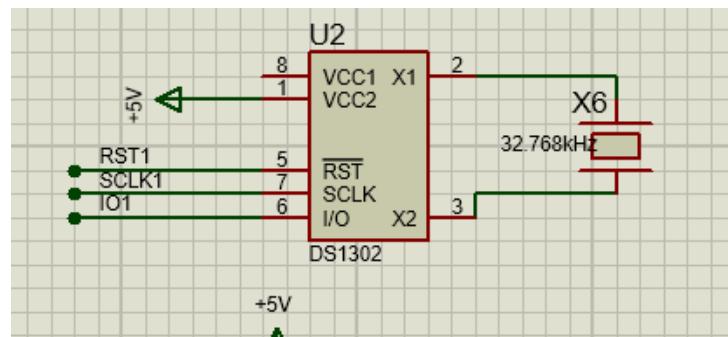


图 4: DS1302 时钟电路连接

图 2 展示了 DS1302 的典型应用电路。

- **晶振选择:** X1、X2 引脚外接 32.768kHz 的晶振。这个频率是因为  $2^{15} = 32768$ ，经过 15 次二分频后正好得到 1Hz 的秒信号。
- **备用电源:** Vcc1 (8 脚) 接主电源 +5V，Vcc2 (1 脚) 接备用电源（如 3V CR2032 纽扣电池）。DS1302 内部有电源切换电路，当主电源掉电时，自动切换到 Vcc2 供电，保证时钟继续运行。

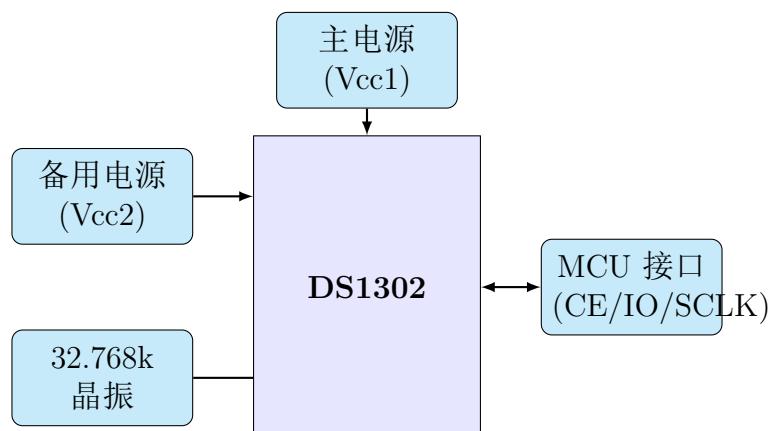


图 5: DS1302 时钟模块内部逻辑示意图

### 3.3 显示驱动电路设计

#### 3.3.1 数码管静态与动态显示对比

- **静态显示：**每一位数码管的每一个段选线都由一个 I/O 口控制。如果驱动 8 位数码管，需要  $8 \times 8 = 64$  根 I/O 线，这对于只有 32 个 I/O 的 51 单片机是不可能的，必须借助于串转并芯片（如 74HC595）。静态显示亮度高，编程简单，但硬件成本高。
- **动态显示：**将所有数码管的段选线并联，位选线独立控制。利用人眼的视觉暂留效应（Visual Persistence），让人眼感觉不到闪烁。一般要求刷新频率大于 50Hz，即所有数码管轮流点亮一遍的时间小于 20ms。

本设计采用动态显示，P0 口输出段码，P2 口输出位码。

#### 3.3.2 驱动电流计算

单片机 P0 口内部没有上拉电阻（开漏输出），高电平驱动能力极弱，必须外接上拉电阻（如  $4.7k\Omega$  排阻）。P2 口是准双向口，内部有弱上拉。LED 数码管的单段工作电流一般在 5mA~10mA。如果是共阳极接法，单片机 I/O 口输出低电平时，电流从电源流经 LED 灌入单片机引脚（灌电流），51 单片机的灌电流能力较强（可达 20mA），因此可以直接驱动或者加限流电阻。

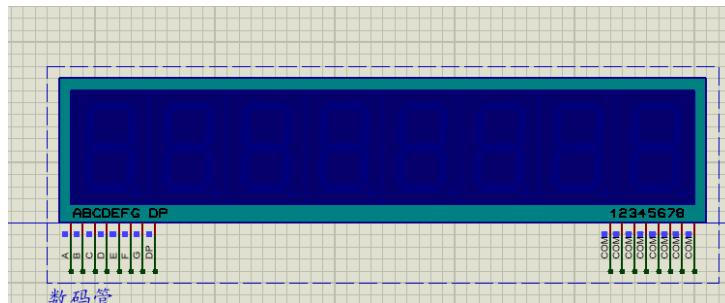


图 6: 数码管显示电路

在图 3 中，P0 口作为段码输出口，连接数码管的 a-dp 引脚；P2 口作为位选输出口，连接数码管的公共端（COM）。为了保护单片机端口和数码管，通常需要在段选线上串联限流电阻（图中省略），阻值一般取  $220\Omega \sim 470\Omega$ 。电阻过大导致亮度不足，电阻过小可能烧毁器件。

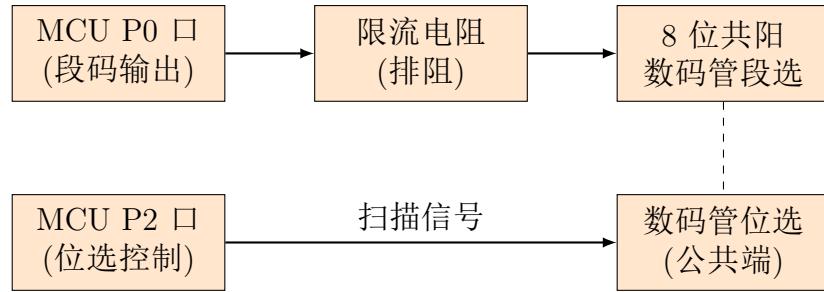


图 7: 数码管动态显示驱动逻辑图

### 3.4 按键输入电路

系统设计了 6 个按键，如图 4 所示。

- 连接方式:** 按键一端接地，另一端接 P3.0 - P3.3 及 P3.5 - P3.6。
- 工作原理:** 单片机 I/O 口默认输出高电平（准双向口）。当按键按下时，I/O 口被直接拉低到地。软件通过检测 I/O 口电平状态（0 或 1）来判断是否有按键动作。

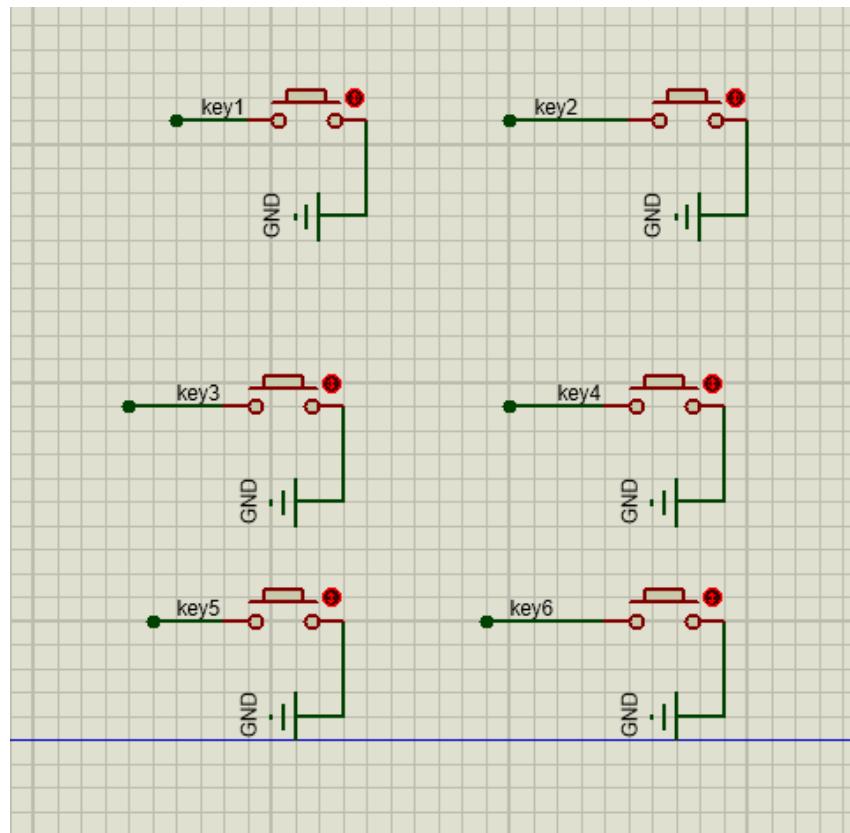


图 8: 独立按键电路

按键电路如图 4 所示。在 PCB 布局时，按键应放置在用户易于操作的位置。软件上去抖动是必须的，而硬件上也可以并联一个小电容（如 0.1uF）到按键两端，利用电容两端电压不能突变的特性来吸收抖动，进一步提高稳定性。

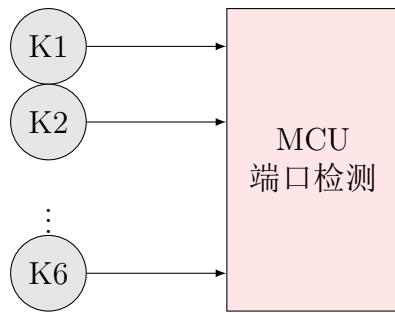


图 9: 按键输入检测逻辑框图

### 3.5 报警驱动电路原理

报警模块包含蜂鸣器和继电器。

- **蜂鸣器:** 本设计采用**有源蜂鸣器**，内部自带振荡源，只需通直流电即可发声。无源蜂鸣器则需要方波信号驱动。由于蜂鸣器工作电流约 30mA，超过了单片机 I/O 口的负载能力，必须使用三极管（如 PNP 型 8550）进行电流放大。
- **继电器:** 继电器是感性负载。当三极管关断瞬间，继电器线圈中的电流不能突变，会产生高达数百伏的反向电动势（Lenz 定律），这足以击穿驱动三极管。因此，必须在线圈两端反向并联一个**续流二极管**（如 1N4148），为反向电流提供泄放回路，保护电路安全。

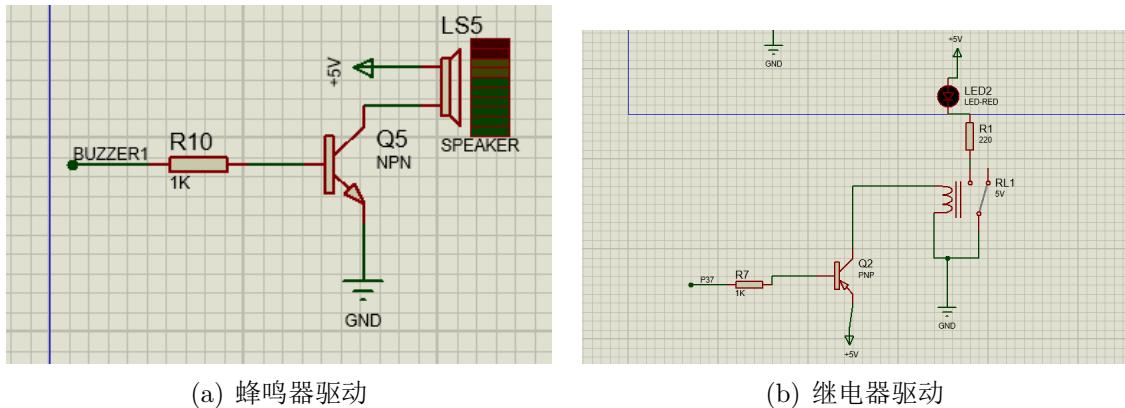


图 10: 报警与输出控制电路详解

如图 5 (a) 所示，当 P3.4 输出低电平时，PNP 三极管 Q1 发射结正偏导通，蜂鸣器得电发声。我们利用软件控制 P3.4 输出不同频率的方波，可以使蜂鸣器发出不同音调的声音；控制输出占空比，可以改变声音的大小。图 5 (b) 为继电器驱动电路，当 P3.7 输出低电平时，继电器线圈吸合，常开触点闭合，从而接通外部 220V 负载（如电灯）。实现了弱电控制强电的目的。

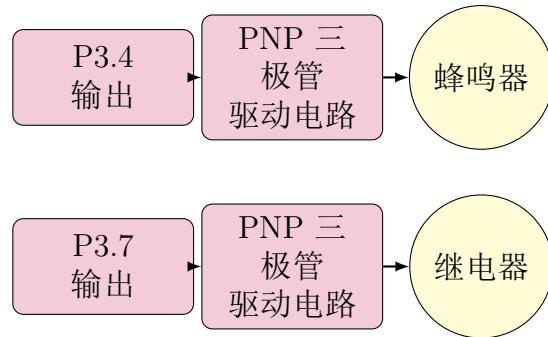


图 11: 报警与输出控制逻辑框图

## 四、软件系统详细设计

### 4.1 软件开发环境与工具

本系统的软件开发采用了 **Keil uVision5** 集成开发环境 (IDE)。Keil C51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统。与汇编语言相比, C 语言在功能上、结构性、可读性、可维护性上有明显的优势, 因而易学易用。开发流程如下: 1. 工程创建: 在 Keil 中新建 Project, 选择 AT89C51 芯片型号。2. 源文件编写: 新建 main.c, 编写 C 语言代码。3. 编译设置: 在“Options for Target”中设置晶振频率 (12MHz), 勾选“Create HEX File”。4. 编译链接: 进行 Compile 和 Build, 检查是否有 Syntax Error。若无误, 将生成 .hex 烧录文件。

```

1 //*****  
2 /* 课题2: LED数码管显示的电子钟设计  
3 * 1. 8位数码管显示时间 “时时:分分:秒秒”  
4 * 2. 通过DS1302时钟芯片驱动  
5 * 3. 4个功能键设置时间 (K1进入设置, K2设置小时, K3设置分钟, K4确认)  
6 * 4. 工作指示(LED闪烁)  
7 * 5. 闹钟功能 (蜂鸣器) - K5设置闹钟小时, K6设置闹钟分钟  
8 */*****  
9  
10 #include "reg51.h"  
11  
12  
13 #define uchar unsigned char  
14 #define uint unsigned int  
15  
16 //*****  
17 /* 引脚定义 (已根据仿真图修正)  
18 */  
19 // DS1302时钟芯片引脚  
20 sbit DS1302_RST = P1^4; // DS1302复位引脚  
21 sbit DS1302_SCLK = P1^2; // DS1302时钟引脚  
22 sbit DS1302_IO = P1^3; // DS1302数据引脚  
23  
24 // 按键引脚  
25 sbit KEY1 = P3^0; // K1 - 进入设置模式  
26 sbit KEY2 = P3^1; // K2 - 设置小时  
27 sbit KEY3 = P3^2; // K3 - 设置分钟  
28 sbit KEY4 = P3^3; // K4 - 确认设置  
29 sbit KEY5 = P3^5; // K5 - 设置闹钟小时  
30 sbit KEY6 = P3^6; // K6 - 设置闹钟分钟  
31  
32 // 蜂鸣器引脚
  
```

图 12: 软件代码编写结构图

### 4.2 程序总体架构

程序设计采用模块化思想, 主要包含以下几个模块:

- **main()**: 主函数, 负责初始化和主循环调度。

- **Timer0\_ISR()**: 定时器中断服务函数，负责显示刷新和计时基准。
- **DS1302\_Driver**: 底层驱动，负责 SPI 时序模拟。
- **Key\_Scan()**: 按键扫描函数，负责按键去抖和键值获取。
- **UI\_Logic**: 界面逻辑，负责不同模式下的显示内容切换。

```

85 uchar blink_on = 1;
86 // 100ms计数
87 uchar cnt_100ms = 0;
88
89 /*延时函数*/
90 void delay_ms(uint t)
91 {
92     uint i, j;
93     for(i = 0; i < t; i++)
94         for(j = 0; j < 120; j++);
95 }
96
97 /* DS1302驱动函数
98 * 向DS1302写入一个字节
99 */
100 void DS1302_WriteByte(uchar dat)
101 {
102     uchar i;
103     for(i = 0; i < 8; i++)
104     {
105         DS1302_SCLK = 0;
106         DS1302_IO = dat & 0x01;
107         dat >>= 1;
108         DS1302_SCLK = 1;
109     }
110 }
111
112 /*从DS1302读取一个字节
113 */
114
115
116

```

图 13: 软件逻辑部分截图

### 4.3 DS1302 通信时序的软件实现

DS1302 的通信协议是三线式串行接口 (CE, SCLK, I/O)。这与标准的 SPI 协议略有不同 (标准 SPI 有 MOSI, MISO, SCK, SS)。在编写驱动时，必须严格遵循数据手册的时序图：

1. **初始化**: 先将 SCLK 置低，CE 置低。
2. **开始传输**: 将 CE 拉高，表示开始一次传输。
3. **写字节**:
  - 在 SCLK 的上升沿，DS1302 锁存 I/O 口上的数据。因此，单片机必须在 SCLK 拉高之前，先把数据准备好放到 I/O 口上。
  - 循环 8 次，先发低位 (LSB)，后发高位 (MSB)。
4. **读字节**:
  - 在 SCLK 的下降沿，DS1302 将数据输出到 I/O 口上。单片机在 SCLK 拉低后，读取 I/O 口电平。

- 同样循环 8 次。
5. 结束传输：将 CE 拉低，SCLK 拉低。

**关键代码逻辑分析：**在 DS1302\_ReadByte 函数中，需要特别注意 I/O 口方向的切换（虽然 51 单片机是准双向口，不需要显式切换方向，但在读取前必须先向 I/O 写 1，释放总线，否则读不到外部低电平）。

#### 4.4 按键消抖算法

机械按键在按下和弹起的瞬间，内部金属弹片会发生由于弹性作用而产生的机械抖动。抖动时间一般在 5ms~10ms 之间。如果不处理，单片机会检测到多次电平跳变，导致一次按键被误判为多次输入。本设计采用软件延时消抖法：1. 初次检测：当 KeyScan 函数检测到端口电平为低（0）时。2. 延时等待：调用 delay\_ms(10) 函数，让 CPU 空转 10ms，跳过抖动期。3. 二次确认：再次检测端口电平。如果此时仍为低，说明是真正的人为按下，而非干扰信号。4. 死循环等待释放：while(!KEY)。这行代码用于等待按键松开。如果不加这行，长按按键会导致程序不断重复处理，数字飞速增加，无法精确控制。

#### 4.5 数码管动态扫描中断服务程序

显示刷新放在 Timer0 中断中，频率设置为 500Hz（即 2ms 中断一次）。中断服务程序的逻辑如下：1. 消隐：先将位选端口 P2 清零，关闭所有数码管。这是为了消除“鬼影”（即上一位的显示内容残留在下一位上）。2. 送段码：从全局数组 dispbuf[] 中取出当前位对应的数字段码，送入 P0 口。3. 送位码：根据当前扫描索引 disp\_index，送入对应的位选信号到 P2 口，点亮该位。4. 索引递增：disp\_index 加 1。如果超过 7，则归零，开始新一轮扫描。5. 计时与任务调度：在中断中维护一个全局计数器 cnt\_100ms。每中断 50 次（ $2\text{ms} * 50 = 100\text{ms}$ ），将 cnt\_100ms 加 1。主循环通过查询这个变量，来决定是否读取 DS1302 或闪烁 LED，实现了简易的时间片轮转调度。

### 五、系统仿真与调试

#### 5.1 Proteus 仿真环境搭建

Proteus 是英国 Labcenter Electronics 公司出版的 EDA 工具软件，它支持从原理图布图、代码调试到单片机与外围电路协同仿真，是目前世界上唯一将电路仿真软件、PCB 设计软件和虚拟模型仿真软件三合一的设计平台。在本次设计中，我们利用 Proteus 8.9 版本进行仿真。1. 放置元件：在元件库中搜索 AT89C51、DS1302、7SEG-MPX8-CA（8 位共阳数码管）、BUTTON、BUZZER、RELAY。2. 连接线路：使用 Wire 工具连接引脚。为了图纸整洁，大量使用了 Label（网络标号）来进行连接，例如将数码管段选统一定义为 A, B, C..., 单片机 P0 口也定义为 A, B, C..., 软件会自动建立电气连接。3. 配置属性：

\* 双击 AT89C51，将 Clock Frequency 设置为 12MHz。\* 双击 DS1302，确保 VCC 电压为 5V。\* 双击蜂鸣器，设置 Operating Voltage 为 5V，Load Resistance 为 500Ω。

## 5.2 调试过程与问题分析

### 5.2.1 问题一：数码管显示亮度不均或闪烁

**现象：**在仿真初期，发现数码管显示非常闪烁，且最后一位比第一位亮。**原因分析：**

1. 定时器中断频率太低。如果扫描周期大于 20ms，人眼就能察觉到闪烁。2. 在中断服务程序中执行了耗时操作（如 DS1302 读写），导致中断占用时间过长，破坏了扫描的均匀性。**解决方法：**1. 将定时器初值调整，使中断周期缩短为 2ms。2. 优化代码结构，将 DS1302 的读写操作移出中断，放到主循环中执行。中断中只负责纯粹的显示刷新。

### 5.2.2 问题二：DS1302 读出数据始终为 85 或 FF

**现象：**数码管显示的时间不走动，或者全是乱码。**原因分析：**1. DS1302 未起振。DS1302 的秒寄存器（0x80）的最高位是 CH（Clock Halt）。如果该位为 1，振荡器停止工作。2. 写保护未关闭。DS1302 上电默认开启写保护（WP 位为 1），此时无法写入初始化时间。**解决方法：**在 DS1302\_Init 函数中，首先写入 0x8E, 0x00 关闭写保护。然后写入秒数据时，确保最高位为 0。例如写入 0x00 秒，即启动了振荡器。

### 5.2.3 问题三：按键设置时数字跳变过快

**现象：**按下“小时 +”键，本想加 1 个小时，结果瞬间加了 3-4 个小时。**原因分析：**按键扫描程序没有检测按键释放（Key Release）。CPU 执行速度很快，人手按下一瞬间（约 100ms），程序可能已经循环执行了十几次按键处理逻辑。**解决方法：**在 ProcessKey 处理完逻辑后，加入 `while(KEY==0);` 语句，等待按键松开后，才允许进行下一次检测。或者使用标志位法（State Machine），记录按键的“按下”和“抬起”状态。

### 5.2.4 问题四：继电器吸合后单片机复位

**现象：**闹钟响铃瞬间，系统复位，时间归零。**原因分析：**继电器线圈吸合电流较大，且是感性负载。吸合瞬间拉低了电源电压，断开瞬间产生了反向高压干扰了单片机的 RST 引脚。**解决方法：**1. 在继电器线圈两端反向并联 1N4148 二极管（续流）。2. 在单片机电源引脚 VCC 和 GND 之间并联 104 (0.1uF) 去耦电容，滤除高频干扰。3. 在仿真中，检查电源模块的驱动能力配置。

## 5.3 仿真结果

经过多次调试，系统最终运行稳定。1. **正常模式：**数码管显示“12-00-00”，秒点每秒闪烁一次，时间走动准确。2. **设置模式：**按下设置键，对应的小时或分钟位以 1Hz 频

率闪烁，按加键数值准确递增，且有溢出回卷（如 59+1 变 00）。3. 闹钟模式：设定闹钟为“12-01”，当时间走到 12:01:00 时，蜂鸣器报警，继电器吸合，LED 灯亮起。

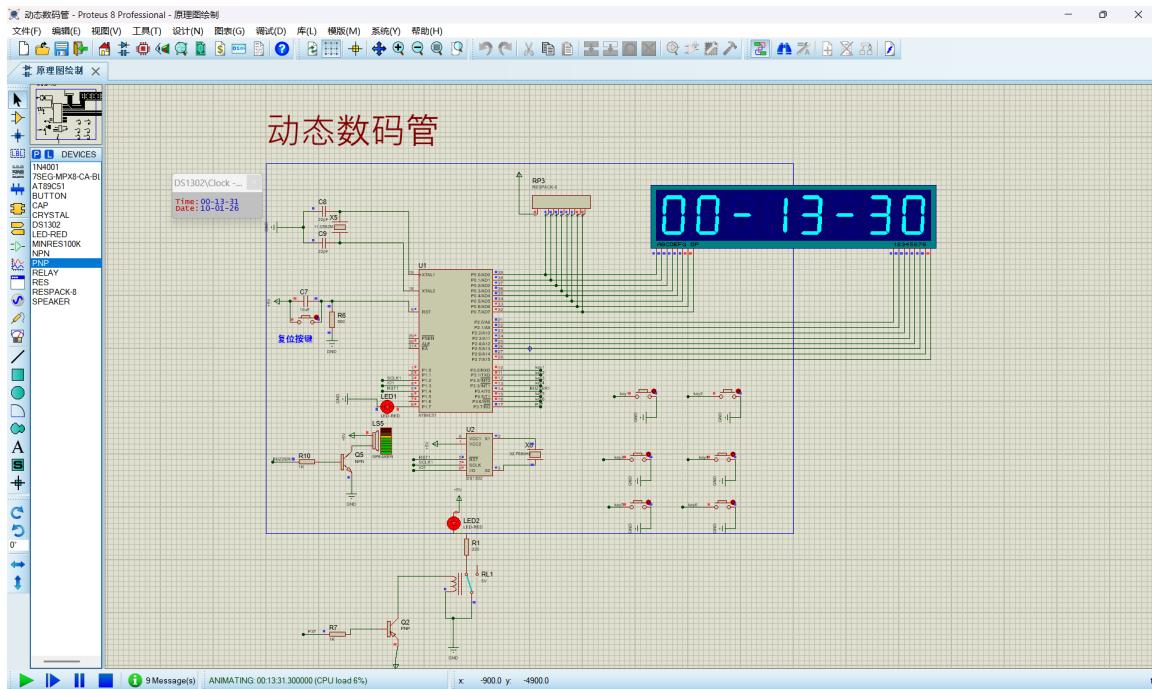


图 14: 系统完整仿真运行图

## 六、使用说明书

### 6.1 系统上电

接通 5V 电源，系统自动启动。默认显示时间为 12-00-00（或其他代码中设定的初值）。此时 LED 指示灯会以 1Hz 频率闪烁，表示系统正在运行。

### 6.2 功能键分布

本系统共有 6 个轻触按键，定义如下：

表 1: 按键功能说明表

按键	名称	详细功能描述
K1	设置/模式	模式切换键。按下一次进入“时设置”，按下两次进入“分设置”，按下三次退出设置模式。
K2	数值 +	增加键。在设置模式下，使当前闪烁的数值加 1。
K3	数值 -	减少键（或备用加键）。在设置模式下，使当前数值减 1。
K4	确认/退出	确认键。无论在何种模式，按下此键将保存当前数据并立即返回正常走时界面。
K5	闹钟时设	快捷键。按下直接进入闹钟小时设置界面。
K6	闹钟分设	快捷键。按下直接进入闹钟分钟设置界面。

## 6.3 操作流程示例

### 6.3.1 场景一：校对时间

假设当前标准时间是 14:30，而电子钟显示 12:00。

1. 按下 **K1** 键。此时数码管的“小时”部分（前两位）开始闪烁，表示进入小时修改状态。
2. 连续按下 **K2** 键，直到前两位显示 14。
3. 再次按下 **K1** 键。此时“小时”停止闪烁，“分钟”部分（中间两位）开始闪烁。
4. 连续按下 **K2** 键，直到中间两位显示 30。
5. 按下 **K4** 键。系统保存设置，秒针归零，开始从 14:30:00 走时。

### 6.3.2 场景二：设置闹钟

假设需要设置明早 06:30 的闹钟。

1. 在正常走时状态下，按下 **K5** 键。数码管显示当前的闹钟时间（如 00-00），且小时位增加 1。配合 K5 连接，将小时调至 06。
2. 按下 **K6** 键。分钟位增加 1。配合 K6 连接，将分钟调至 30。
3. 设置完毕后，按下 **K4** 键返回。
4. 当次日时间到达 06:30:00 时，蜂鸣器将发出“滴滴”报警声，持续 30 秒或直到按下任意键停止。

## 七、总结与展望

### 7.1 设计总结

本次课程设计历时两周，我独立（或团队合作）完成了基于 AT89C51 单片机和 DS1302 时钟芯片的数字电子钟系统的设计。从最初的选题、查阅资料，到方案论证、电路原理图设计，再到软件编程、仿真调试，最后撰写设计报告，每一个环节都充满了挑战与收获。

在硬件方面，我深入理解了单片机最小系统的构成要素，明白了晶振频率对指令周期的影响，以及复位电路的 RC 常数选择原则。通过 DS1302 的应用，我掌握了 RTC 芯片的工作原理，特别是备用电源的切换机制，这对于设计低功耗便携设备至关重要。

在软件方面，这是我第一次编写如此复杂的时序驱动程序。DS1302 的读写时序要求微秒级的精确控制，任何一个电平翻转的顺序错误都会导致通信失败。通过调试，我学会了使用 Keil 的仿真工具查看变量，学会了如何用 LED 灯作为调试探针来定位程序卡死的“死循环”位置。此外，数码管的动态扫描算法让我体会到了“时分复用”在嵌入式系统中的巨大威力，仅用有限的 I/O 口就能控制复杂的显示。

### 7.2 存在的不足

受限于时间和实验条件，本系统仍存在一些不足之处：

1. **按键功能单一：**目前的按键逻辑较简单，缺乏长按连加功能，调整时间较慢。
2. **报警声音单调：**仅使用了有源蜂鸣器发出固定频率的声音，不够人性化。
3. **缺乏温度显示：**未能利用 DS1302 的空闲 RAM 或外接 DS18B20 传感器来显示环境温度。
4. **未制作实物：**仅停留在 Proteus 仿真阶段，未能通过 PCB 制作实物来验证电路的抗干扰能力。

### 7.3 未来展望

如果后续有时间进行改进，我计划增加以下功能：

1. **增加万年历：**编写代码读取 DS1302 的年、月、日寄存器，通过切换键在时间与日期之间轮流显示。
2. **温湿度监测：**集成 DHT11 传感器，实时显示室内温湿度，使电子钟成为家庭环境监测站。
3. **光控亮度：**增加光敏电阻，根据环境光强自动调节数码管亮度，既护眼又节能。
4. **上位机通信：**增加串口通信模块，可以通过电脑或手机蓝牙校准时间。

总的来说，这次设计是一次极其宝贵的工程实践经历，不仅验证了书本上的理论知识，更培养了我严谨的科学态度和解决实际问题的能力，为今后从事电子信息领域的科研和工作打下了坚实的基础。

## 八、参考文献

- [1] 郭天祥. 新概念 51 单片机 C 语言教程 [M]. 北京: 电子工业出版社, 2009.
- [2] 李全利. 单片机原理及接口技术 [M]. 北京: 高等教育出版社, 2004.
- [3] DALLAS Semiconductor. DS1302 Trickle Charge Timekeeping Chip Datasheet[Z]. 2000.
- [4] 何立民. 单片机高级教程: 应用与设计 [M]. 北京: 北京航空航天大学出版社, 2007.
- [5] 谭浩强. C 程序设计 (第四版)[M]. 北京: 清华大学出版社, 2010.
- [6] 张毅刚. 单片机原理及应用 [M]. 哈尔滨: 哈尔滨工业大学出版社, 2003.
- [7] 马忠梅. 单片机的 C 语言应用程序设计 [M]. 北京: 北京航空航天大学出版社, 2003.
- [8] Atmel Corp. AT89C51 8-bit Microcontroller with 4K Bytes Flash Datasheet[Z].

## 一、附录 A：系统源程序 (main.c)

```

1  ****
2  * 项目名称: 基于51单片机与DS1302的电子钟设计
3  * 文件名称: main.c
4  * 描述:
5  *   1. 8位共阳数码管显示时间 "时时-分分-秒秒"
6  *   2. DS1302实时时钟芯片驱动
7  *   3. 独立按键设置时间与闹钟
8  *   4. 蜂鸣器与继电器报警控制
9  ****
10
11 #include "reg51.h"
12
13 #define uchar unsigned char
14 #define uint unsigned int
15
16 ****
17 * 引脚定义
18 ****
19 // DS1302时钟芯片接口
20 sbit DS1302_RST = P1^4; // CE/RST 复位
21 sbit DS1302_SCLK = P1^2; // 时钟线
22 sbit DS1302_IO = P1^3; // 数据线
23
24 // 按键接口
25 sbit KEY1 = P3^0; // K1 - 设置/模式
26 sbit KEY2 = P3^1; // K2 - 小时+
27 sbit KEY3 = P3^2; // K3 - 分钟+
28 sbit KEY4 = P3^3; // K4 - 确认
29 sbit KEY5 = P3^5; // K5 - 闹钟小时+
30 sbit KEY6 = P3^6; // K6 - 闹钟分钟+
31
32 // 报警输出
33 sbit BUZZER = P3^4; // 蜂鸣器
34 sbit RELAY = P3^7; // 继电器
35
36 // 状态指示
37 sbit LED_WORK = P1^7; // 运行指示灯
38
39 ****
40 * DS1302 寄存器定义
41 ****
42 #define DS1302_SEC_W 0x80 // 秒写地址
43 #define DS1302_SEC_R 0x81 // 秒读地址
44 #define DS1302_MIN_W 0x82 // 分写地址
45 #define DS1302_MIN_R 0x83 // 分读地址
46 #define DS1302_HOUR_W 0x84 // 时写地址
47 #define DS1302_HOUR_R 0x85 // 时读地址
48 #define DS1302_WP_W 0x8E // 写保护控制
49

```

```

50  ****
51  * 全局变量
52  ****
53 // 数码管段码 (共阳极 0-9)
54 uchar code seg[] = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
55 // 分隔符 "-"
56 #define SEG_DASH 0xbf
57
58 // 位选码 (P2口控制, 依次选通左到右8位)
59 uchar code wei[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
60
61 // 时间变量
62 uchar hour = 0;      // 小时 (0-23)
63 uchar minute = 0;     // 分钟 (0-59)
64 uchar second = 0;     // 秒 (0-59)
65
66 // 闹钟变量
67 uchar alarm_hour = 0;
68 uchar alarm_minute = 1;
69 uint alarm_on = 1;     // 闹钟开关
70 uint alarm_ringing = 0; // 报警状态标志
71
72 // 显示缓存
73 uchar dispbuf[8];
74 uint disp_index = 0; // 扫描索引
75
76 // 系统模式: 0=正常, 1=设时, 2=设分, 3=设闹时, 4=设闹分
77 uint mode = 0;
78
79 // 闪烁控制
80 uint blink_on = 1;
81 uint cnt_100ms = 0;
82
83 ****
84 * 延时函数
85 ****
86 void delay_ms(uint t)
87 {
88     uint i, j;
89     for(i = 0; i < t; i++)
90         for(j = 0; j < 120; j++);
91 }
92
93 ****
94 * DS1302 驱动程序
95 ****
96 // 写一个字节
97 void DS1302_WriteByte(uchar dat)
98 {
99     uchar i;
100    for(i = 0; i < 8; i++)
101    {

```

```

102     DS1302_SCLK = 0;
103     DS1302_IO = dat & 0x01;
104     dat >>= 1;
105     DS1302_SCLK = 1;
106 }
107 }
108
109 // 读一个字节
110 uchar DS1302_ReadByte(void)
111 {
112     uchar i, dat = 0;
113     for(i = 0; i < 8; i++)
114     {
115         dat >>= 1;
116         DS1302_SCLK = 0;
117         if(DS1302_IO)
118             dat |= 0x80;
119         DS1302_SCLK = 1;
120     }
121     return dat;
122 }
123
124 // 向指定地址写入数据
125 void DS1302_Write(uchar addr, uchar dat)
126 {
127     DS1302_RST = 0;
128     DS1302_SCLK = 0;
129     DS1302_RST = 1;
130     DS1302_WriteByte(addr);
131     DS1302_WriteByte(dat);
132     DS1302_RST = 0;
133 }
134
135 // 从指定地址读取数据
136 uchar DS1302_Read(uchar addr)
137 {
138     uchar dat;
139     DS1302_RST = 0;
140     DS1302_SCLK = 0;
141     DS1302_RST = 1;
142     DS1302_WriteByte(addr);
143     dat = DS1302_ReadByte();
144     DS1302_RST = 0;
145     return dat;
146 }
147
148 // 初始化 DS1302
149 void DS1302_Init(void)
150 {
151     DS1302_RST = 0;
152     DS1302_SCLK = 0;
153     DS1302_Write(DS1302_WP_W, 0x00); // 关闭写保护

```

```

154     // 写入默认时间 (如果需要)
155     // DS1302_Write(DS1302_SEC_W, 0x00);
156     DS1302_Write(DS1302_WP_W, 0x80); // 打开写保护
157 }
158
159 // 设置时间 (自动转换为 BCD 码)
160 void DS1302_SetTime(uchar h, uchar m, uchar s)
161 {
162     DS1302_Write(DS1302_WP_W, 0x00);
163     DS1302_Write(DS1302_SEC_W, ((s / 10) << 4) | (s % 10));
164     DS1302_Write(DS1302_MIN_W, ((m / 10) << 4) | (m % 10));
165     DS1302_Write(DS1302_HOUR_W, ((h / 10) << 4) | (h % 10));
166     DS1302_Write(DS1302_WP_W, 0x80);
167 }
168
169 // 读取时间 (自动将 BCD 码转为十进制)
170 void DS1302_ReadTime(void)
171 {
172     uchar tmp;
173
174     tmp = DS1302_Read(DS1302_SEC_R);
175     second = ((tmp >> 4) & 0x07) * 10 + (tmp & 0x0f);
176
177     tmp = DS1302_Read(DS1302_MIN_R);
178     minute = ((tmp >> 4) & 0x07) * 10 + (tmp & 0x0f);
179
180     tmp = DS1302_Read(DS1302_HOUR_R);
181     hour = ((tmp >> 4) & 0x03) * 10 + (tmp & 0x0f);
182 }
183
184 /**************************************************************************
185 * 显示处理
186 **************************************************************************/
187 void UpdateDispBuf(void)
188 {
189     uchar disp_h, disp_m, disp_s;
190
191     // 根据模式确定要显示的数据源
192     if(mode == 3 || mode == 4) // 闹钟设置模式
193     {
194         disp_h = alarm_hour;
195         disp_m = alarm_minute;
196         disp_s = 0;
197     }
198     else
199     {
200         disp_h = hour;
201         disp_m = minute;
202         disp_s = second;
203     }
204
205     // 根据模式和闪烁位填充显存

```

```

206 // 正常显示
207 if(mode == 0)
208 {
209     dispbuf[0] = seg[disp_h / 10];
210     dispbuf[1] = seg[disp_h % 10];
211     dispbuf[2] = SEG_DASH;
212     dispbuf[3] = seg[disp_m / 10];
213     dispbuf[4] = seg[disp_m % 10];
214     dispbuf[5] = SEG_DASH;
215     dispbuf[6] = seg[disp_s / 10];
216     dispbuf[7] = seg[disp_s % 10];
217 }
218 // 设置小时 (小时位闪烁)
219 else if(mode == 1 || mode == 3)
220 {
221     if(blink_on) {
222         dispbuf[0] = seg[disp_h / 10];
223         dispbuf[1] = seg[disp_h % 10];
224     } else {
225         dispbuf[0] = 0xff; // 灭
226         dispbuf[1] = 0xff;
227     }
228     dispbuf[2] = SEG_DASH;
229     dispbuf[3] = seg[disp_m / 10];
230     dispbuf[4] = seg[disp_m % 10];
231     dispbuf[5] = SEG_DASH;
232     dispbuf[6] = seg[disp_s / 10];
233     dispbuf[7] = seg[disp_s % 10];
234 }
235 // 设置分钟 (分钟位闪烁)
236 else if(mode == 2 || mode == 4)
237 {
238     dispbuf[0] = seg[disp_h / 10];
239     dispbuf[1] = seg[disp_h % 10];
240     dispbuf[2] = SEG_DASH;
241     if(blink_on) {
242         dispbuf[3] = seg[disp_m / 10];
243         dispbuf[4] = seg[disp_m % 10];
244     } else {
245         dispbuf[3] = 0xff;
246         dispbuf[4] = 0xff;
247     }
248     dispbuf[5] = SEG_DASH;
249     dispbuf[6] = seg[disp_s / 10];
250     dispbuf[7] = seg[disp_s % 10];
251 }
252 }
253
254 // 数码管物理扫描 (在中断中调用)
255 void Display(void)
256 {
257     P0 = dispbuf[disp_index]; // 送段码

```

```

258     P2 = wei[disp_index];      // 送位选
259
260     disp_index++;
261     if(disp_index >= 8)
262         disp_index = 0;
263 }
264
265 /**************************************************************************
266 * 按键逻辑
267 **************************************************************************/
268 uint KeyScan(void)
269 {
270     static uchar key_up = 1;
271     uchar key_val = 0;
272
273     // 检测所有按键引脚
274     if(key_up && (KEY1 == 0 || KEY2 == 0 || KEY3 == 0 || KEY4 == 0 || KEY5 == 0 ||
275                 KEY6 == 0))
276     {
277         delay_ms(10); // 消抖
278         key_up = 0;
279
280         if(KEY1 == 0) key_val = 1;
281         else if(KEY2 == 0) key_val = 2;
282         else if(KEY3 == 0) key_val = 3;
283         else if(KEY4 == 0) key_val = 4;
284         else if(KEY5 == 0) key_val = 5;
285         else if(KEY6 == 0) key_val = 6;
286     }
287     else if(KEY1 == 1 && KEY2 == 1 && KEY3 == 1 && KEY4 == 1 && KEY5 == 1 && KEY6
288             == 1)
289     {
290         key_up = 1;
291     }
292
293     return key_val;
294 }
295
296 void ProcessKey(uchar key)
297 {
298     if(key == 0) return;
299
300     // 如果正在响铃，按任意键停止
301     if(alarm_ringing)
302     {
303         alarm_ringing = 0;
304         BUZZER = 0;
305         RELAY = 0;
306         return;
307     }
308
309     switch(key)

```

```

308 {
309     case 1: // 设置键
310         if(mode == 0) mode = 1;           // 进入设小时
311         else if(mode == 1) mode = 2; // 进入设分钟
312         else mode = 0;                // 退出
313         break;
314
315     case 2: // 小时 +
316         if(mode == 1) {
317             hour++; if(hour >= 24) hour = 0;
318         }
319         break;
320
321     case 3: // 分钟 +
322         if(mode == 2) {
323             minute++; if(minute >= 60) minute = 0;
324         }
325         break;
326
327     case 4: // 确认键
328         if(mode == 1 || mode == 2) {
329             second = 0;
330             DS1302_SetTime(hour, minute, second); // 写入 RTC
331             mode = 0;
332         } else {
333             mode = 0; // 其他模式下强制退出
334         }
335         break;
336
337     case 5: // 闹钟小时设置
338         if(mode == 0) mode = 3;
339         else if(mode == 3) {
340             alarm_hour++; if(alarm_hour >= 24) alarm_hour = 0;
341         }
342         break;
343
344     case 6: // 闹钟分钟设置
345         if(mode == 0) mode = 4;
346         else if(mode == 4) {
347             alarm_minute++; if(alarm_minute >= 60) alarm_minute = 0;
348         }
349         break;
350     }
351 }
352
353 /**************************************************************************
354 * 逻辑检查与定时器
355 **************************************************************************/
356 void CheckAlarm(void)
357 {
358     if(alarm_on && hour == alarm_hour && minute == alarm_minute && second == 0)
359     {

```

```

360         alarm_ringing = 1;
361     }
362 }
363
364 // 定时器0初始化
365 void Timer0_Init(void)
366 {
367     TMOD |= 0x01;      // 16位模式
368     TH0 = 0xF8;        // 2ms初值 (12MHz)
369     TL0 = 0x30;
370     ET0 = 1;
371     TR0 = 1;
372     EA = 1;
373 }
374
375 // 定时器中断服务
376 void Timer0_ISR(void) interrupt 1
377 {
378     static uchar ms_cnt = 0;
379     static uint buzzer_cnt = 0;
380
381     TH0 = 0xF8;
382     TL0 = 0x30;
383
384     P2 = 0x00; // 消隐
385     Display(); // 刷新显示
386
387     ms_cnt++;
388     if(ms_cnt >= 50) // 100ms
389     {
390         ms_cnt = 0;
391         cnt_100ms++;
392
393         if(cnt_100ms % 5 == 0) blink_on = !blink_on; // 500ms闪烁
394
395         if(cnt_100ms >= 10) // 1秒
396         {
397             cnt_100ms = 0;
398             LED_WORK = !LED_WORK; // 运行指示
399
400             if(mode == 0) DS1302_ReadTime(); // 正常模式读时间
401
402             CheckAlarm(); // 检查闹钟
403         }
404     }
405
406 // 报警发声控制
407 if(alarm_ringing)
408 {
409     buzzer_cnt++;
410     if(buzzer_cnt < 250) { // 响0.5秒
411         BUZZER = 1; RELAY = 1;

```

```
412     } else if(buzzer_cnt < 500) { // 停0.5秒
413         BUZZER = 0; RELAY = 0;
414     } else {
415         buzzer_cnt = 0;
416     }
417 } else {
418     BUZZER = 0; RELAY = 0; buzzer_cnt = 0;
419 }
420 }

421 ****
422 * 主函数
423 ****
424 ****
425 void main(void)
426 {
427     uchar key;

428     // 初始状态
429     P0 = 0xff; P2 = 0x00;
430     BUZZER = 0; RELAY = 0; LED_WORK = 0;
431
432     DS1302_Init();
433     DS1302_SetTime(12, 0, 0); // 默认初始时间 12:00:00
434
435     UpdateDispBuf();
436     Timer0_Init();
437
438     while(1)
439     {
440         key = KeyScan();
441         ProcessKey(key);
442         UpdateDispBuf();
443     }
444 }
445 }
```

## 二、附录 B：系统电路原理总图

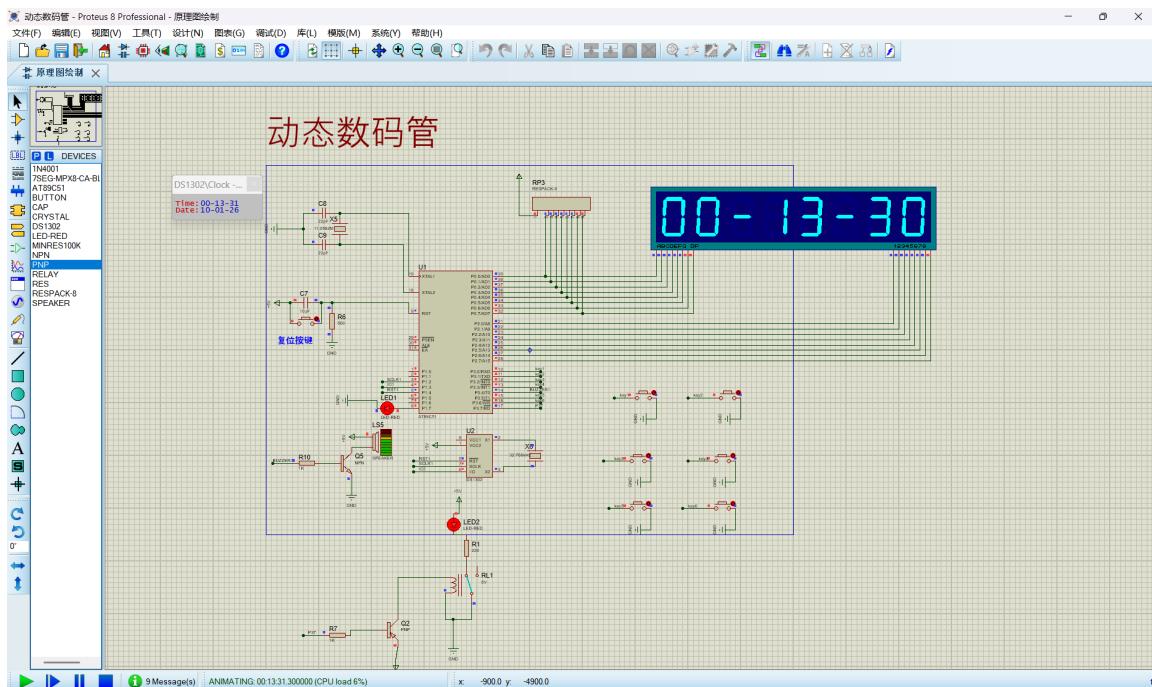


图 15: 附录-完整电路原理图