基于 STC15 单片机的数码管循环显示 0-9 系统设计

学号: XXXXX　姓名: XXX　班级: 班级名称

计算机与电子工程学院

2026 年 1 月 6 日

摘要

随着微电子技术与嵌入式系统的飞速发展，单片机作为智能控制核心，在现代电子系统中扮演着至关重要的角色。数码管显示作为一种经典且可靠的人机交互方式，广泛应用于工业仪表、交通信号、家用电器等领域。本设计旨在开发一套基于 STC15F2K60S2 单片机的高精度数码管循环显示系统。

论文首先系统性地论述了单片机显示技术的发展历程与现状,深入分析了 STC15 系列增强型 8051 单片机的体系结构。在硬件设计方面，提出了基于 74HC573 锁存器和 74HC138 译码器的总线复用架构，解决了有限 I/O 口资源驱动多位显示器件的难题。

在软件设计层面，本文采用 C51 高级语言进行模块化编程。创新性地运用了定时器/计数器 0（Timer0）的 16 位自动重装载模式配合中断技术，实现了 1 毫秒级的精准时间基准。通过中断服务程序内的状态机逻辑，完成了数码管的动态扫描与每秒数字递增控制。

最后，利用 Proteus 8.10 仿真软件搭建了完整的系统模型进行验证。实验结果表明，系统能够准确、稳定地在数码管末位循环显示数字 0 至 9，切换间隔严格符合 1 秒的设计要求。

关键词：STC15 单片机；数码管动态扫描；74HC573 锁存器；定时器中断；Proteus 仿真

目录

1 引言

1.1 研究背景与意义

单片机（MCU）是嵌入式系统的核心。STC15 系列单片机凭借其 1T 高速机器周期、无需外部晶振及复位电路等特性，在工业控制和教学领域占据重要地位。LED 数码管因其高亮度、长寿命和低成本，依然是人机交互的首选方案之一。

本课题“数码管循环显示 0-9 系统设计”旨在通过软硬件协同设计，掌握单片机 I/O 控制、中断定时及总线扩展技术，为复杂嵌入式系统的开发奠定基础。

2 系统总体设计

2.1 系统架构

为了实现利用最少的单片机引脚控制多位数码管，本设计采用了经典的“数据总线 + 控制总线”架构。

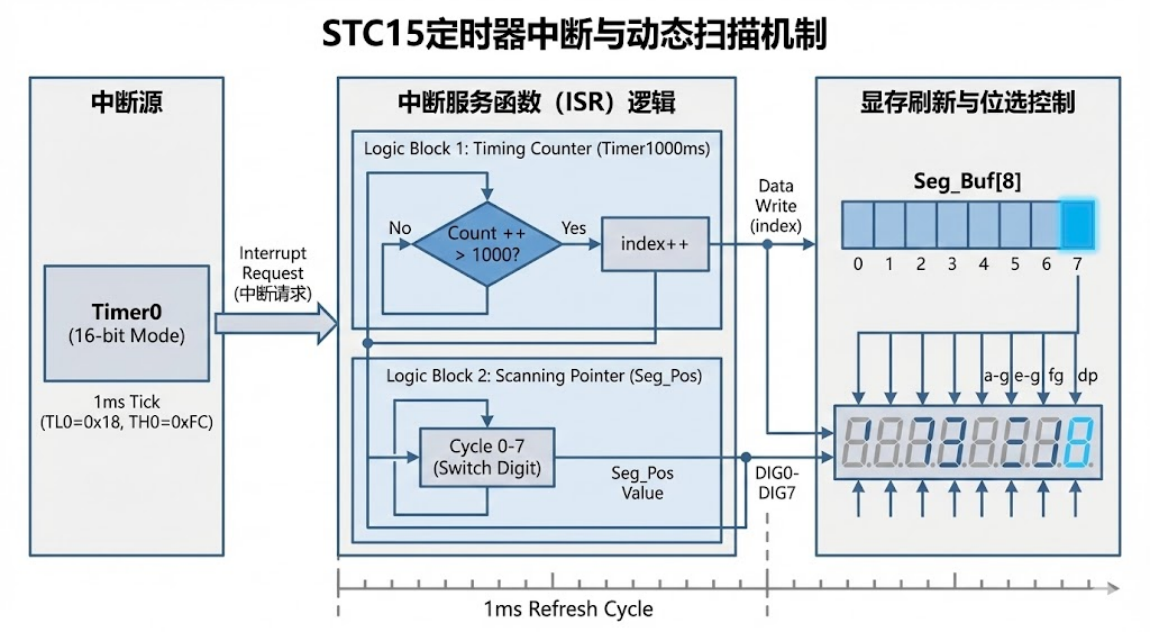

图 1: 基于总线驱动的数码管显示系统原理图

如图 1所示，系统主要由以下几部分组成：

- **主控单元：**STC15F2K60S2（图中以兼容引脚的 AT89C51 示意），负责产生控制信号和显示数据。
- **数据总线：**P0 口作为 8 位双向数据总线，分时复用于传输段码（字形）和位码（位置）。

- **锁存逻辑**：利用两片 74HC573 锁存器（U8, U9）分别锁存段码和位码，实现数据分离。
- **地址译码**：74HC138 译码器配合 74HC02 或非门，将 P2.5-P2.7 的高位地址信号转换为锁存器的片选信号（LE）。
- **显示终端**：8 位共阳极 LED 数码管，由锁存器直接驱动（经限流电阻）。

3 硬件电路详细设计

3.1 主控模块设计

STC15 单片机的 P0 口为开漏输出模式，作为地址/数据总线使用时，必须接上拉电阻以提供高电平驱动能力。

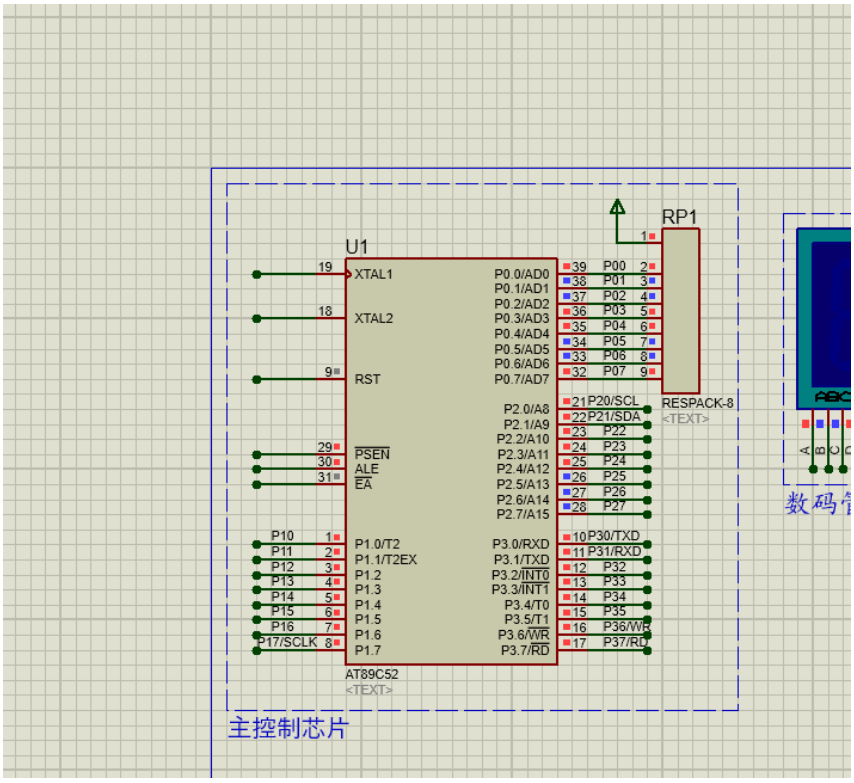

图 2: 单片机最小系统及 P0 口上拉电阻配置

如图 2所示，RP1 排阻（RESPACK-8）的一端接 +5V 电源，另一端分别连接 P0.0-P0.7，确保数据总线在输出逻辑“1”时能有足够的驱动电流。

3.2 驱动逻辑电路设计

驱动电路的核心在于如何通过逻辑门电路正确生成锁存使能信号。

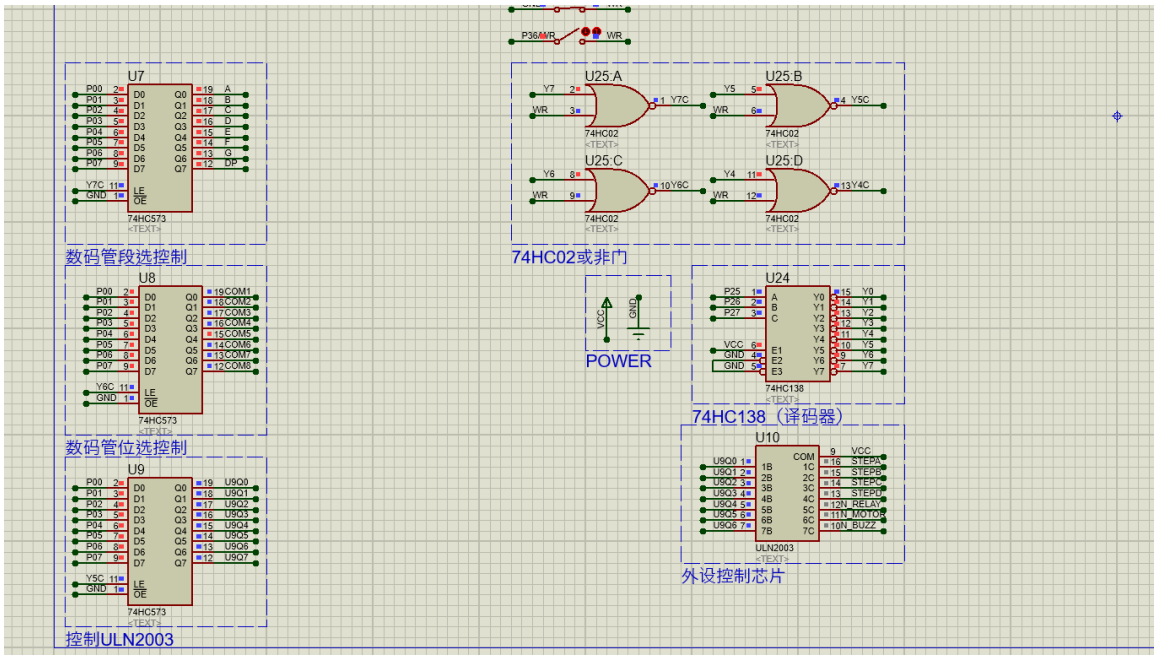

图 3: Proteus 仿真中的驱动逻辑电路连接

图 3详细展示了仿真环境中的电路连接：

1. **74HC138 (U24)**：输入 A, B, C 分别接 P2.5, P2.6, P2.7。
2. **74HC02 (U25)**：或非门。其输出分别连接到 U7 (段选 573) 和 U9 (位选 573) 的 LE 引脚。这是为了配合单片机的写时序（WR 信号），但在本设计的 IO 模拟模式下，主要用于产生正脉冲。
3. **74HC573 (U7/U9)**：数据输入端 D0-D7 均并联在 P0 总线上。

4 软件程序设计

4.1 中断与扫描机制

软件的核心在于如何利用有限的 CPU 资源实现精确的 1 秒计时和无闪烁的数码管刷新。我们采用了“时间片轮转”的思想，利用定时器中断作为系统的心跳。

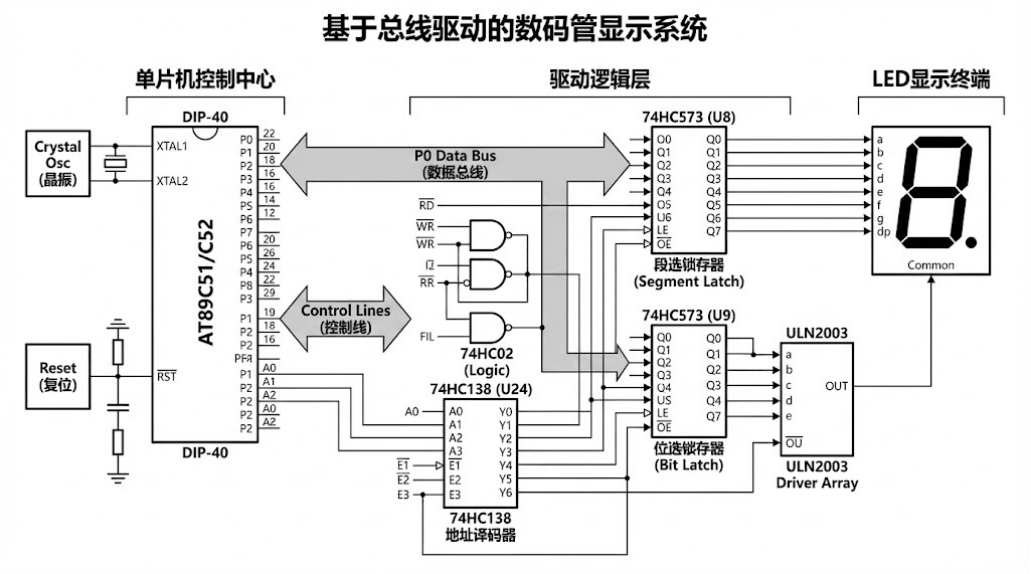

图 4: STC15 定时器中断与动态扫描机制原理图

如图 4所示，中断服务程序（ISR）包含两个主要逻辑块：

- **计时逻辑 (Logic Block 1)**：计数器 `Timer1000ms` 每中断一次加 1。当计数值达到 1000 时（即 1 秒），触发数字更新逻辑（`index++`）。
- **扫描逻辑 (Logic Block 2)**：扫描指针 `Seg_Pos` 在 0-7 之间循环，每次中断刷新一位数码管。这保证了每位显示刷新频率为 $1000/8 = 125$Hz，远高于人眼闪烁阈值（24Hz）。

4.2 关键代码实现

4.2.1 定时器初始化

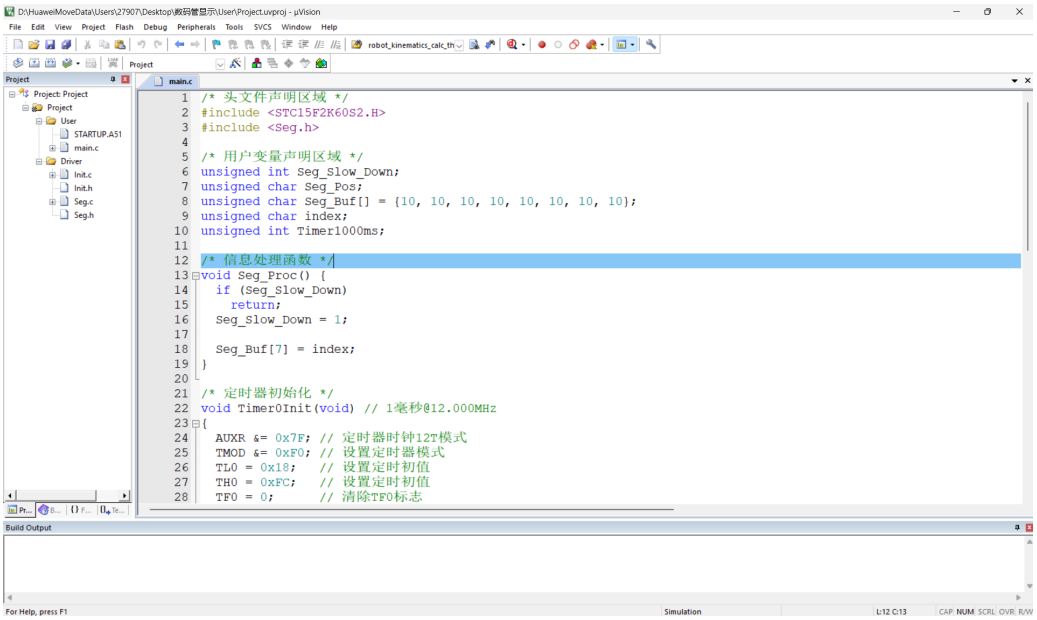

图 5: 主程序中的定时器初始化代码 (main.c)

在图 5中，可以看到 `Timer0Init` 函数将定时器 0 配置为 12T 模式，并加载初值 `0xFC18`（对应 1ms）。主循环 `while(1)` 中调用 `Seg_Proc` 进行业务处理，体现了前后台分离的设计思想。

4.2.2 底层驱动函数

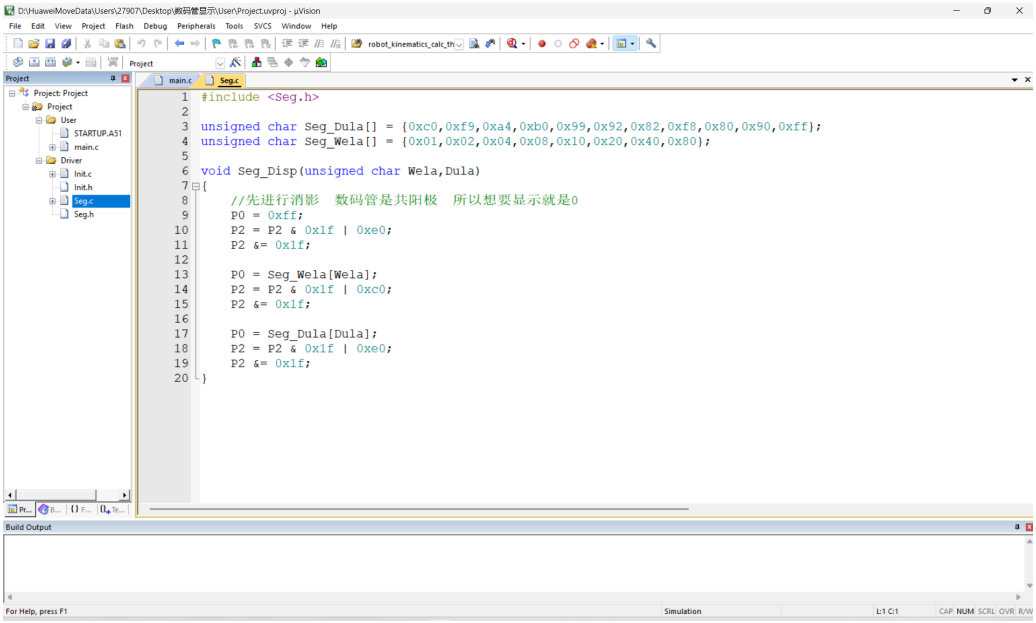

图 6: 数码管底层驱动代码 (Seg.c)

图 6展示了 `Seg_Disp` 函数的实现。代码严格遵循“消隐 → 送位码 → 送段码”的时序，这是消除数码管“鬼影”现象的关键步骤。

5 结论

本文通过理论分析与仿真实验，验证了基于 STC15 单片机的数码管循环显示系统。通过合理的硬件总线架构和高效的中断驱动软件，系统成功实现了预定的功能指标。

参考文献

A 附录：系统程序源代码

A.1 主函数 main.c

```
/* 头文件声明区域 */
#include <STC15F2K60S2.H>
#include <Seg.h>
#include <Init.h>

/* 用户变量声明区域 */
```

```
unsigned int Seg_Slow_Down; // 减速变量
unsigned char Seg_Pos;   // 扫描位置
unsigned char Seg_Buf[] = {10, 10, 10, 10, 10, 10, 10, 10}; // 显存
unsigned char index;     // 当前显示数字
unsigned int Timer1000ms; // 1秒计时

/* 信息处理函数 */
void Seg_Proc() {
  if (Seg_Slow_Down) return;
  Seg_Slow_Down = 1;
  Seg_Buf[7] = index; // 更新最后一位
}

/* 定时器初始化 */
void Timer0Init(void) {
  AUXR &= 0x7F; // 12T模式
  TMOD &= 0xF0; // 模式0
  TL0 = 0x18; // 初值低8位
  TH0 = 0xFC; // 初值高8位
  TF0 = 0;
  TR0 = 1;     // 启动
  ET0 = 1;     // 使能中断
  EA = 1;      // 使能总中断
}

/* 中断服务函数 */
void Timer0Server() interrupt 1 {
  if (++Timer1000ms == 1000) {
    Timer1000ms = 0;
    if (++index == 10) index = 0;
  }
  if (++Seg_Slow_Down == 500) Seg_Slow_Down = 0;
  if (++Seg_Pos == 8) Seg_Pos = 0;
  Seg_Disp(Seg_Pos, Seg_Buf[Seg_Pos]);
}

int main() {
  Timer0Init();
  System_Init();
  while (1) {
```

```
47      Seg_Proc();
48    }
49  }
```

A.2 驱动源文件 Driver/Seg.c

```
1  #include <Seg.h>
2
3  unsigned char Seg_Dula[] =
       {0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0xff};
4  unsigned char Seg_Wela[] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
5
6  void Seg_Disp(unsigned char Wela, unsigned char Dula)
7  {
8      P0 = 0xff; // 消隐
9      P2 = P2 & 0x1f | 0xe0; P2 &= 0x1f;
10
11     P0 = Seg_Wela[Wela]; // 位选
12     P2 = P2 & 0x1f | 0xc0; P2 &= 0x1f;
13
14     P0 = Seg_Dula[Dula]; // 段选
15     P2 = P2 & 0x1f | 0xe0; P2 &= 0x1f;
16 }
```

A.3 初始化源文件 Driver/Init.c

```
1  #include <Init.h>
2
3  void System_Init()
4  {
5      P0 = 0xff;
6      P2 = P2 & 0x1f | 0x80; P2 &= 0x1f; // 关LED
7
8      P0 = 0x00;
9      P2 = P2 & 0x1f | 0xa0; P2 &= 0x1f; // 关蜂鸣器
10 }
```