SST 单片机中文教程

(非 SST 官方资料,请以 SST 网站上 DATASHEET 为准)

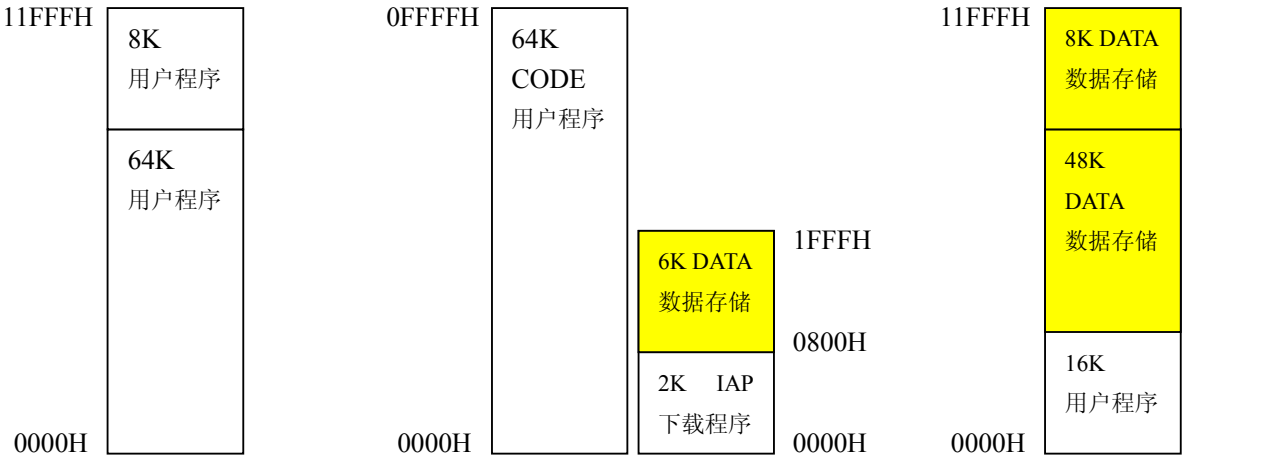
香港弘微科技有限公司

祖微电子产品(上海)有限公司

SST89 系列单片机可实现的功能配置:

1. **片内用户程序空间可达 72K.**
2. **片内 EEPROM 数据存储容量可超 64K.**
3. **5 个通道的 PWM 信号输出,可实现 5 路的 D/A 数模转换.**
4. **6 个 UART 串口.**让产品的通讯功能更加灵活,省掉昂贵的串口扩展芯片
5. **1 个 SPI 串口.**
6. **内嵌电压检测电路,**节省外部的电源管理及复位芯片.
7. **在片仿真功能,**SOFTICE 功能,让开发工程师省掉仿真器,并弥补了专用仿真器的”不能仿真扩展功能,接触不良,编程不能运行,价格昂贵”的缺陷.
8. **在线编程功能.**EASYIAP 工具软件,让开发工程师省掉编程器.
9. **程序和数据存储空间互补利用,**用户程序剩下的 FLASH 空间,均可作为数据存储.超级灵活

SST MCU 内部 FLASH 存储空间的三种用法示意图



第一种用法:
72K 的用户程序空间

第二种用法:
64K 的用户程序空间,
6K DATA 数据存储
2K IAP 下载程序

第三种用法:
16K 的用户程序空间
其余空间 58K 作数据存储

超高性价比,超高灵活性的 SST89 系列单片机

弘微科技有限公司（SPAC）

超高性价比:

1. **节省成本:**
   1. **节省系统成本:** 可省去外部的 EEPROM(最大容量达 64K,相当于 28/24C512,成本约 20 元),电压监控芯片,上电复位电路(约 2 元),5 个 UART 的扩展芯片(约 80 元)
   2. **节省开发成本:** 自带在线下载和在片仿真功能,省去了开发用的编程器和仿真器(约 8000 元)
   3. **节省生产成本:** 一个专用编程芯片搞定芯片的生产编程,节省了生产部门配置的电脑,生产编程器及其适配座(约 10000 元), 简单,安全,保密,易于生产管理
   4. **节省维护成本:** 具有客户端的产品升级功能,可迅速解决客户现场出现的开发过程中没有遇到的特殊问题,节省产品的邮寄费用,维护人员的飞机出差费用,并不耽误客户的产品使用.让客户感觉你的产品具有无限更新的功能,对开发留下的 BUG 问题可以忽略不计,提高了企业的声誉.如果按照常规的返修流程,影响了客户的使用,以后可能就不会再买你的产品了.这个成本是无法用金钱来衡量的.
2. **方便开发:** 具有 SOFTICE 在片仿真功能,开发工程师不需仿真器和编程器便可开发调试产品.可实现单步、断点、全速的仿真，变量数据一目了然。并且在片仿真解决了专用仿真器存在的-“不能仿真非标准 52 的特殊功能;接触不良;编程不能运行”的缺陷.
3. **方便生产.** 芯片的批量编程,仅需一个芯片搞定.既不需生产部配备电脑,编程器,编程插座等生产设备,又避免了因为人为因素的选错文件,程序泄密等弊端. 简单,安全,保密,易于生产管理
4. **数据存储:** 非易失数据存储（可超过 64K），掉电后数据不丢失，片内数据存储更保密。
5. **应用灵活:** 内嵌两块小扇区（128 字节/扇区）的 FLASH 存储器，程序和数据存储空间充分互补利用， 不作程序的空间,均可作为数据存储.突破了标准 51 单片机的程序与数据必须分开和 64K 的极限.用户程序最大可达 72K,数据存储可超 64K.
6. **最多可达 6 个 UART 串口.** 通讯功能更加灵活方便.
7. **售出产品的产品升级更新功能.** 产品快速上市,可根据客户的反馈及时更新产品,让产品在市场上的始终处于领先地位,增强产品的竞争优势.与客户保持良好的长期合作关系.
8. **防盗版解密功能.** 由于灵活的产品更新和软硬加密功能,可有效防止你的产品解密盗版.保护你的知识权益.

SST89 系列单片机选型一览表

型 号	最高时钟频率Hz		Flash 存储器	RAM	串口 UART	PCA	中断源	优先级	DPTR 数据指针	降低 EMI	掉电检测	看门狗	双倍速	P4 口	SPI
	5V /E	3V /V													
SST89E/V516RD2	40M	33M	64K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V58RD2	40M	33M	32K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V54RD2	40M	33M	16K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V52RD2	40M	33M	8K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V54RC	33M	25M	16K+1K	512B	1ch+	0	8	4	2	✓	✓	✓	✓		✓
SST89E/V52RC	33M	25M	8K+1K	512B	1ch+	0	8	4	2	✓	✓	✓	✓		✓

附：SST 系列单片机性能

SST 单片机

主要特性：

- 兼容 80C51 系列，内置超级 FLASH 存储器的单片机
- SST89E5XXRD
  工作电压 V_{DD}=4.5～5.5V
  5 伏工作电压时 0～40MHz 的频率范围
- SST89V5XXRD
  工作电压 V_{DD}=2.7～3.6V
  在 3 伏工作电压下,原厂保证 0～25 MHz 的工作频率 ,实际最高可达 40MHz
- 与现行的 80C52 列单片机硬件 PIN-TO-PIN 完全兼容，软件、开发工具也完全兼容
- 1K*8 的内部 RAM（256Bytes+768Bytes，可放心使用 C 语言编程）
- 两块超级 FLASH EEPROM
  —SST89E516RD/SST89V516RD:64K*8 的基本存储块和 8K*8 的二级存储块（扇区大小为 128 字节）
  —SST89E58RD/SST89V58RD:32K*8 的基本存储块和 8K*8 的二级存储块（扇区大小为 128 字节），（ 二级存储块可用于存放掉电后要保存的数据，放在内部具有极强的抗干扰性）
  —独立的块加密
  —IAP 下的并行操作
  —块地址重映射
- 最大片外程序/数据地址空间为 64K*8（当然也可以通过 I/O 口进行块切换，实现超 64K 扩展）
- 三个高电流驱动引脚（每个 16 mA,可直接驱动 LED）
- 三个 16 位定时器/计数器
- 全双工增强型 UART
  —帧错误检测
  —自动地址识别
- 9 个中断源，四个中断优先级
- 看门狗定时器（Watchdog Timer, 缺省情况下不打开，用户不需要时可不使用）
- 可编程计数阵列（PCA）
  标准为每个机器周期 12 个时钟，器件可选择在每个机器周期 6 个时钟基础上加倍
  掉电检测（Brow-out 缺省为产生复位，也可设置为产生中断）
- 降低 EMI 模式（通过 AUXR SFR 不允许 ALE 输出时钟）
  以上三项确保了 SST 单片机的高抗干扰性，可直接取代 ATMEL 公司的单片机
- 四个 8 位 I/O 口（32 根输入输出线）

- 双 DPTR 指针（查表，寻址更方便）
- SPI 串行接口
- 兼容 TTL 和 CMOS
- 扩展省电模式
  —Idle 模式
  —由外部中断唤醒的省电模式
  —Standby 模式
- 3 种封装形式：PDIP -40、PLCC-44 、TQFP- 44
- 温度范围：
  商业级（0℃-+70℃ ）
  工业级（-40℃-+85℃）

产品简介：

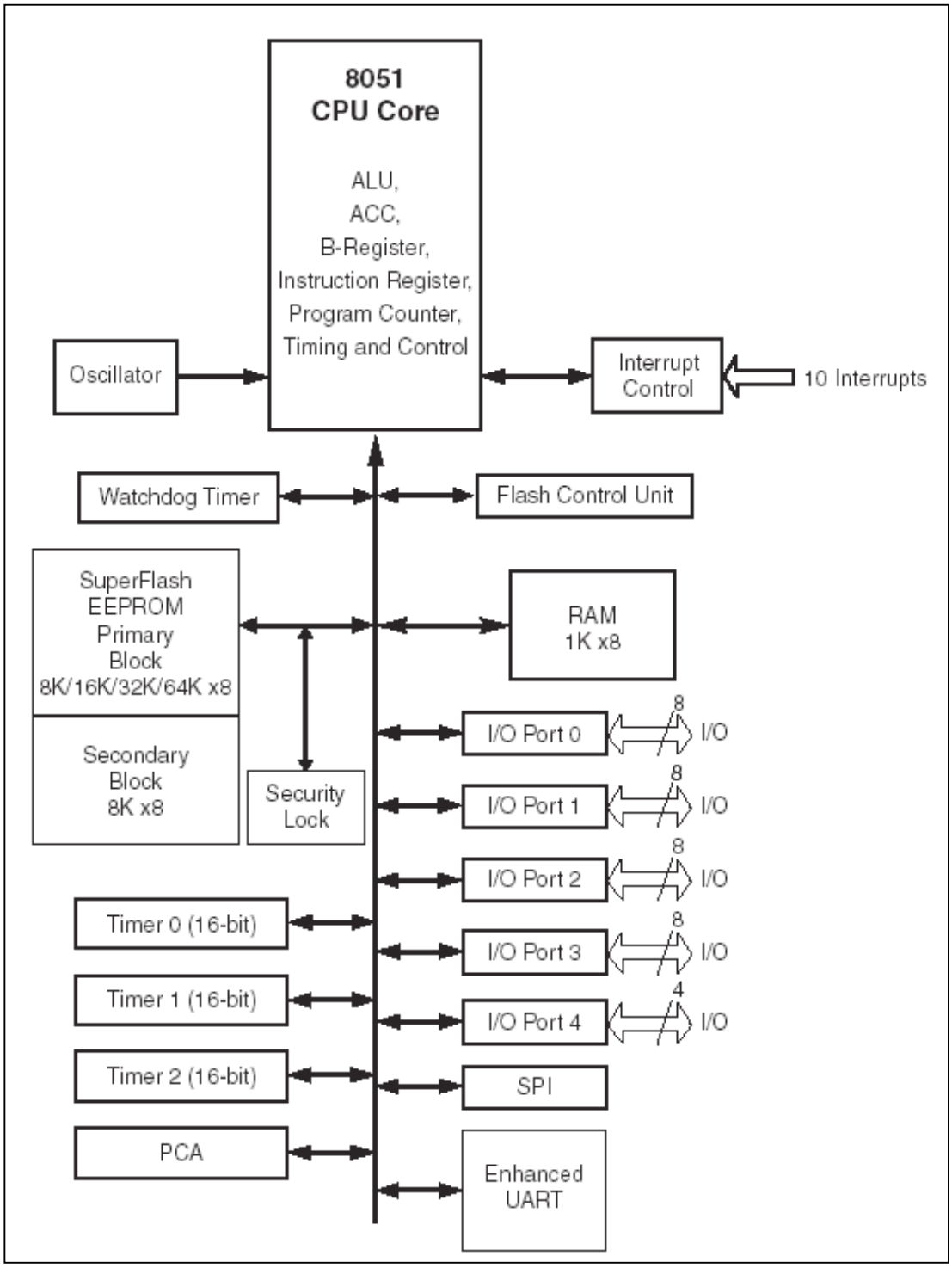
SST89E516RD、SST89V516RD、SST89E58RD 和 SST89V58RD 都是 8 位 FLASH FLEX51 系列单片机。FLASH FLEX51 是在高级 FLASH CMOS 半导体工艺下设计和生产出的单片机产品之一。器件都有相同的功能强大的指令系统，并且和 8xC5x 器件兼容。

单片机有 72/40K 片内 FLASH EEPROM 程序存储器，它利用了 SST 的超级 FLASH 专利技术，这些都是 SST 的领先技术。超级 FLASH 存储器被分成两个独立的程序存储块，基本 FLASH Block0 占用 64/32K 字节片内程序存储空间，二级 FLASH Block 1 占用 8K 字节的片内程序存储空间；8K 字节的二级 FLASH 块能被映射到 64/32K 字节低地址空间它也能从程序计数器中被隐藏掉而用做一个独立的类似 EEPROM 的数据存储器。

FLASH 存储通过标准的 87C5x OTP EPROM 编程器来编程，这个编程器必须有和 SST 器件配套的适配器和硬件。在上电复位过程中，单片机能初始化为一个存储源代码的外部主机的从机，或用来控制外部主机的 IAP 操作。单片机可预先设计存储器内的引导装入程序，可以指导初学者装入程序代码和熟练者通过 IAP 更新代码。引导装入程序仅仅是一个参考并带来方便。单片机并没保证引导装入程序例子的可行性和有用性。芯片擦除和块擦除操作将擦除预先编辑的例子代码。除了 72/40K 字节的超级 FLASH 程序存储器,器件能寻址到 64K 字节外部程序存储空间.除了 1024*8 位的内部 RAM，外部 64K RAM 地址空间也能被寻址.SST 系列单片机的高可靠性,享有专利的超级 FLASH 技术和存储单元结构在设计和制造 MCU 方面有较大的优势, 这些优势产生的高性能价格比，极大的方便了我们的客户。

第 一 章 SST 系列单片机原理

1.1 功能框图



1.2 引脚分配

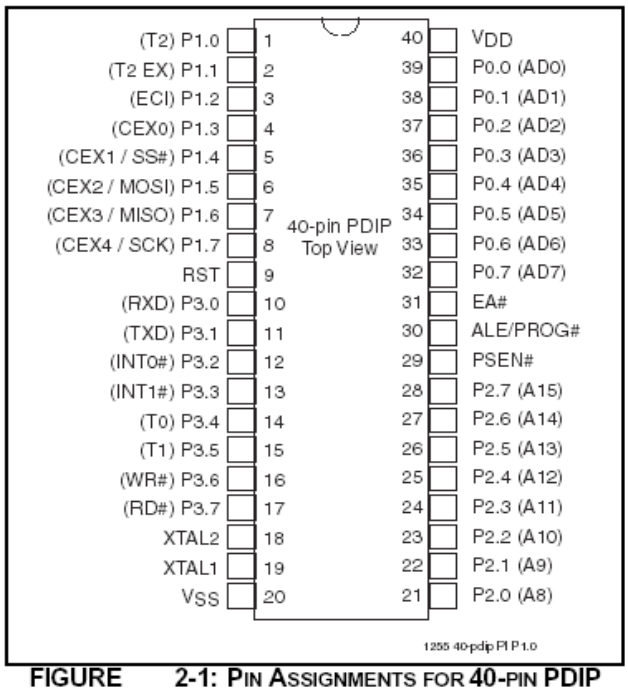


FIGURE 2-1: PIN ASSIGNMENTS FOR 40-PIN PDIP

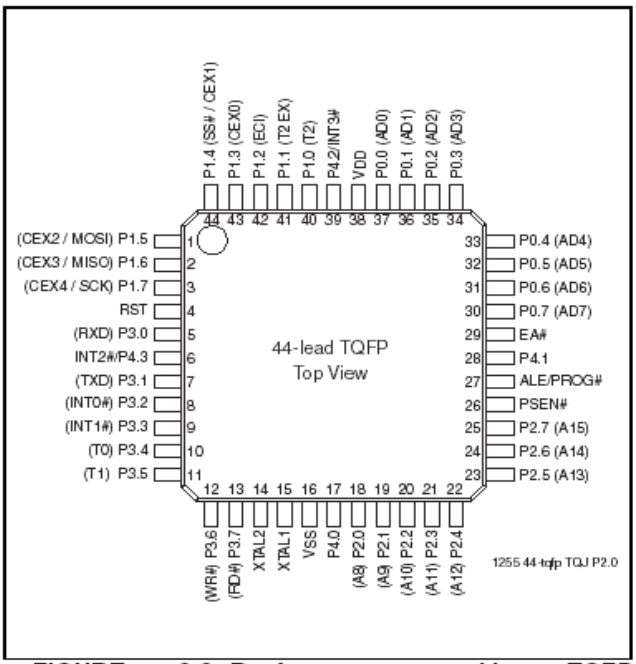
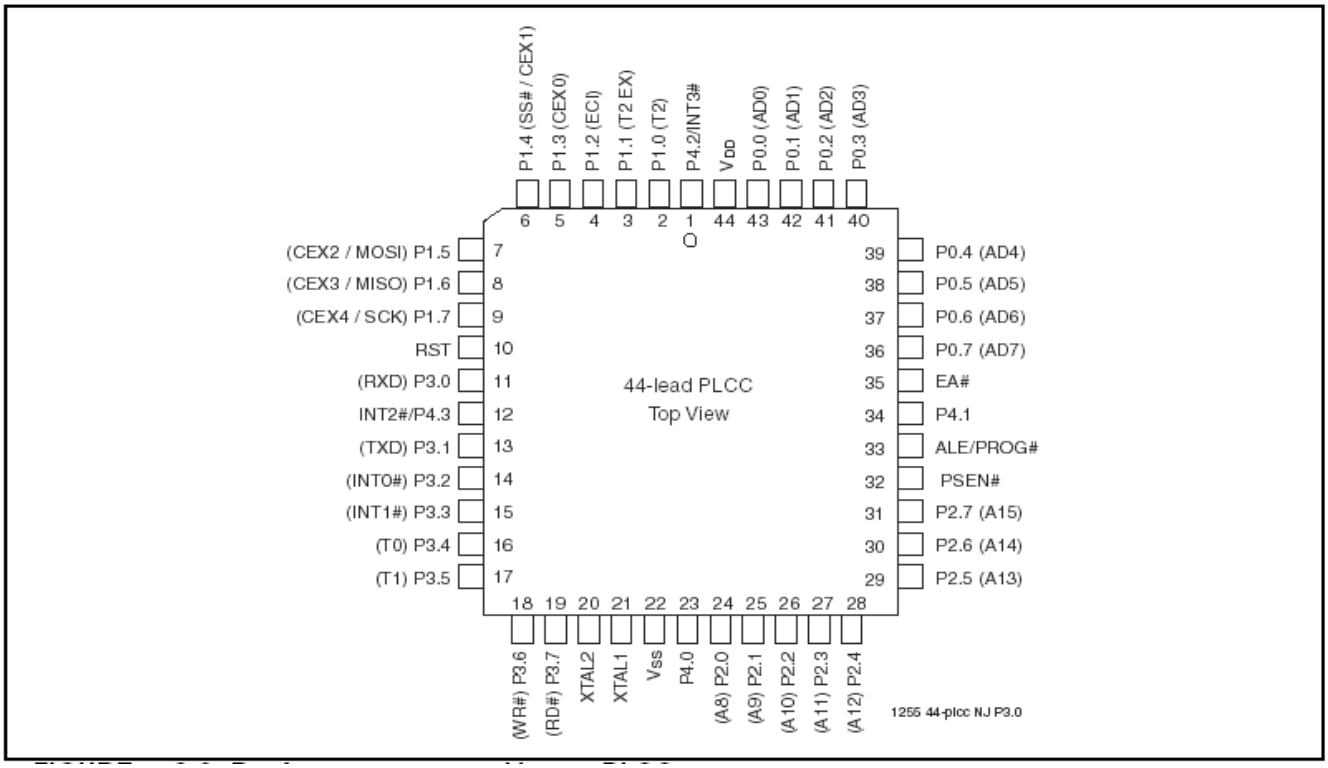


FIGURE 2-2: PIN ASSIGNMENTS FOR 44-LEAD TQFP



1.2 引脚描述

标志	类型	名称和功能
P0[7:0]	I/O	Port 0: P0 是一个漏极开路的 8 位双向 I/O 口。作为输出口，每位能驱动 多个 LS 型 TTL 负载。 P0 浮空，锁存器为“1”，可作为高阻抗输入。 在访问外部存储器时，P0 口作为低 8 位地址和数据总线分时复用。在这种应用中,当转为高电平时,它用了强大的内部上拉。在外部主模式编程状态下,P0 接收代码字节,在外部主模式校验过程中输出代码字节。在程序校验过程中需要外部上拉
P1[7:0]	带内部上拉的 I/O	Port 1：P1 是一个带内部上拉电阻的 8 位准双向 I/O 口。每位能驱动 LS 型 TTL 负载。当 P1 口作为输入口用时，向内部锁存器写入“1”，P1 引脚被内部上拉电阻拉为高电平。由于内部上拉电阻，被拉为低的 P1 引脚能向外提供电流（I_{IL}，如 10-3 和 10-4 图示）。P1[5，6，7] 有 16 毫安的高电流驱动能力。当外部主模式在编程和测试时，P1 也接收低 8 位地址
P1[0]	I/O	T2 ：定时器 /计数器 2 外部计数输入或时钟输出从定时器/计数器 2
P1[1]	I	T2EX： 定时器/计数器 2 捕捉/重装触发器和方向控制
P1[2]	I	EC1：PCA 定时器/计数器外部输入
P1[3]	I/O	CEX0：比较/捕捉外部输入输出模块，每个比较/捕捉模块连接到一 P1 口引脚，当不用于 PCA 时， 这个口用作标准 I/O
P1[4]	I/O	SS#： 主机输入、从机输出（SPI）或 CEX1：比较/捕捉外部输入输出模块
P1[5]	I/O	MOSI：主机输出，从机输入（SPI）或 CEX2：比较/捕捉外部输入输出模块
P1[6]	I/O	MISO：主机输入，从机输出（SPI）或 CEX3：比较/捕捉外部输入输出模块
P1[7]	I/O	SCK： 主机时钟输出、从机时钟输入或 CEX4：比较/捕捉外部输入输出模块
P2[7：0]	带内部上拉的 I/O	Port 2： P2 是一个带内部上拉电阻的 8 位准双向 I/O 端口，当被作为输入时，向它写“1”，P2 引脚被内部上拉电阻拉为高电平。作为输入使用时，被内部上拉电阻下拉为低电平的 P2 会产生电流（I_{IL} 如 10-3 和 10-4 表所示）。当从片外程序存储器取数和访问片外数据存储器时，P2 能提供高 8 位地址。在此应用中，当转为 V_{OH} 时，它利用了功能极强的内部上拉电阻。当外部主模式在编程和测试时，它还接收控制信号和部分高 8 位地址。
P3[7：0]	带内部上拉的 I/O	Port 3： P3 是一个带内部上拉电阻的 8 位准双向 I/O 口。P 3 的输出缓冲能驱动多个 LS 型 TTL 。当被作为输入时，向它写“1”，PORT 3 引脚被内部上拉电阻拉为高电平，作为输入使用时，被外部拉为低，能驱动电流（I_{IL} 如 10-3 和 10-4 表所示）。当外部主机在编程和校验时，它还能接收控制信号和部分高 8 位地址。
P3[0]	I	RXD ：串行数据接收
P3[1]	O	TXD： 串行数据发送
P3[2]	I	INT0#：外部中断 0 输入
P3[3]	I	INT1#：外部中断 1 输入
P3[4]	I	T0：定时/计数器 0 的外部计数输入
P3[5]	I	T1：定时/计数器 1 的外部计数输入
P3[6]	O	WR#：外部数据存储器写选通
P3[7]	O	RD#：外部数据存储器读选通

表 9-2-1 :引脚描述(2)

标志	类型	名称和功能
P4[7：0]	带内部上拉的I/O	Port 3： P3 是一个带内部上拉电阻的 4 位准双向 I/O 口。P 3 的输出缓冲能驱动多个 LS 型 TTL 。当被作为输入时，向它写“1”，PORT 3 引脚被内部上拉电阻拉为高电平，作为输入使用时，被外部拉为低，能驱动电流（I_{IL} 如 10-3 和 10-4 表所示）。
P4[0]	I	RXD ：串行数据接收
P4[1]	O	TXD： 串行数据发送
P4[2]	I	INT3#：外部中断 3 输入
P4[3]	I	INT2#：外部中断 2 输入
PSEN#	I/O	程序存储器允许：PSEN#是外部程序存储器读选通。当从内部程序存储器执行时，PSEN#不激活。当从外部程序存储器执行时，每个机器周期 PSEN#两次有效，除了当进入外部数据存储器时，在每个机器周期都有一个 PSEN#信号跳过。当 RST 输入能持续保持高电平多于 10 个机器周期时，迫使 PSEN#由低到高的转换会使单片机进入主编程模式
RST	I	Reset：振荡器在工作时，此脚如能保持两个机器周期以上的高电平复位器件。复位后，当 RST 输入保持高电平，PSEN#引脚被高到低的电平转换驱动，器件将进入外部主模式，否则，器件将进入通用操作模式
EA#	I	外部访问允许：为了使单片机能从片外程序存储器取指令，EA#必为低。内部程序执行时，EA#必为高电平。然而，第四级加密锁将禁止 EA#，程序只能从片内程序存储器开始执行。EA#能承受 12V 的高电压。（可参考 47 页“绝对最大承受值 ）
ALE/ PROG	I/0	地址允许：在访问外部存储器时，ALE 用于锁存出现在 P0 口的低 8 位地址。此引脚也是外部主模式编程脉冲输入端（PROG#）。除了访问外部数据存储器，ALE 在每个机器周期有效两次，在第二个机器周期有一个 ALE 有效被跳过。然而，如果 AO 置“1“，ALE 被禁止。（可参考 20 页”辅助寄存器“）
XTAL1/	I/0	振荡器：输入输出来转换振荡放大器。XTAL 1 是内部时钟产生电路的输入从外部时钟源
V_{DD}	输入	电源输入：通用、IDLE、省电、备用模式下的电源供应
V_{SS}	输入	接地端：电源接地端（参考电压为 0V）

I = 输入 ； O = 输出 在 Flash 编程期间，并不需要 12V 的电压供应。

1.3 存储器结构

本系列单片机程序和数据存储空间分开独立寻址。

1.3.1 程序存储器

有两个片内 FLASH 存储块。基本 FLASH 存储块（Block 0）占用 64/32/16K 字节，二级 FLASH 存储块 Block 1 为 8K 字节(4K,89C54/58)。因为整个内部程序空间被限制为 64/32K 字节，SFCF[1：0]位用于控制存储块的切换。请参考图 9-3-1 和 9-3-2 中关于程序存储空间的分配。程序存储块的选择下一节再说明。

64/32K *8 的基本超级 FLASH 块被组织成 512/256 个扇区，每个扇区由 128 字节组成。8K *8 二级 FLASH 存储块被组织成 64 个扇区，每个扇区是 128 字节。对两个块来说，程序地址空间的低 7 位选择扇区内的字节，剩下的地址位选择块内扇区。

图：9-3-1： SST89E/V516RD 和 SST89V516RD 的程序存储器组织

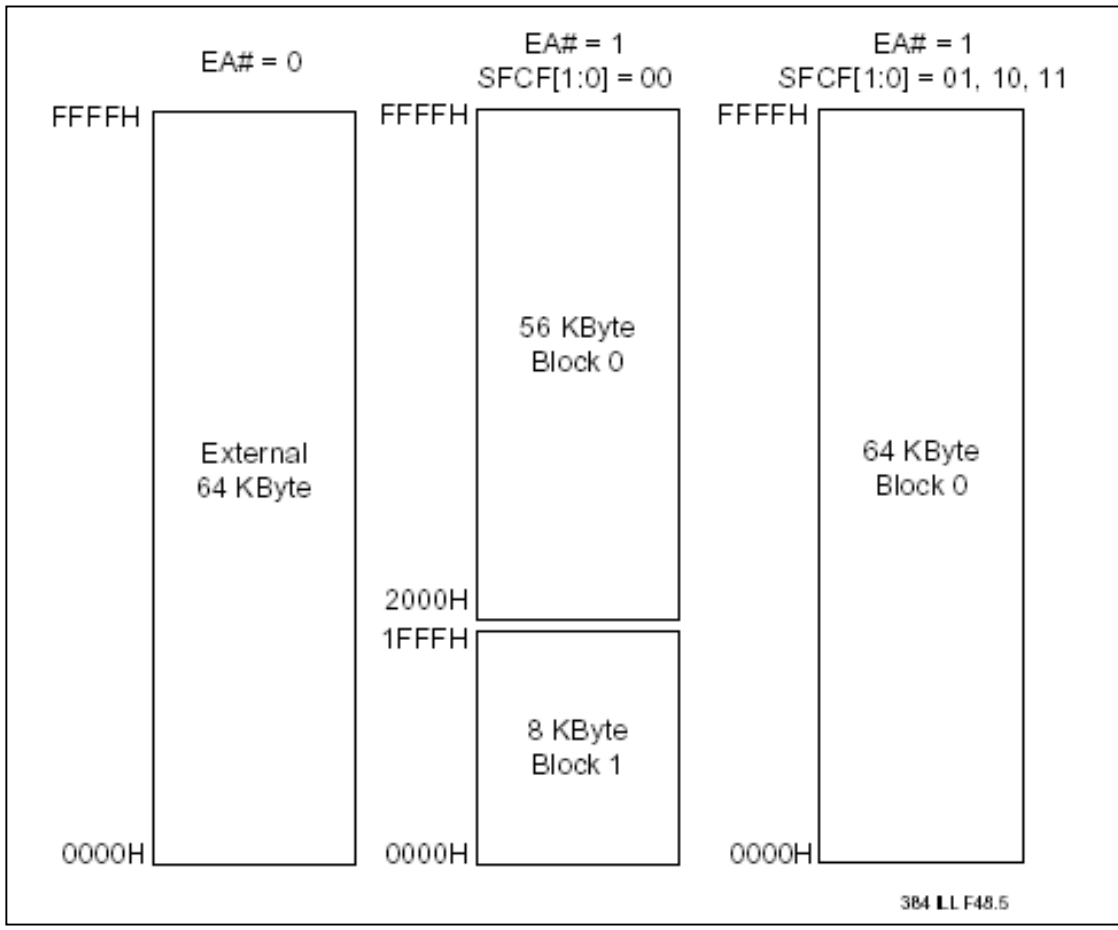
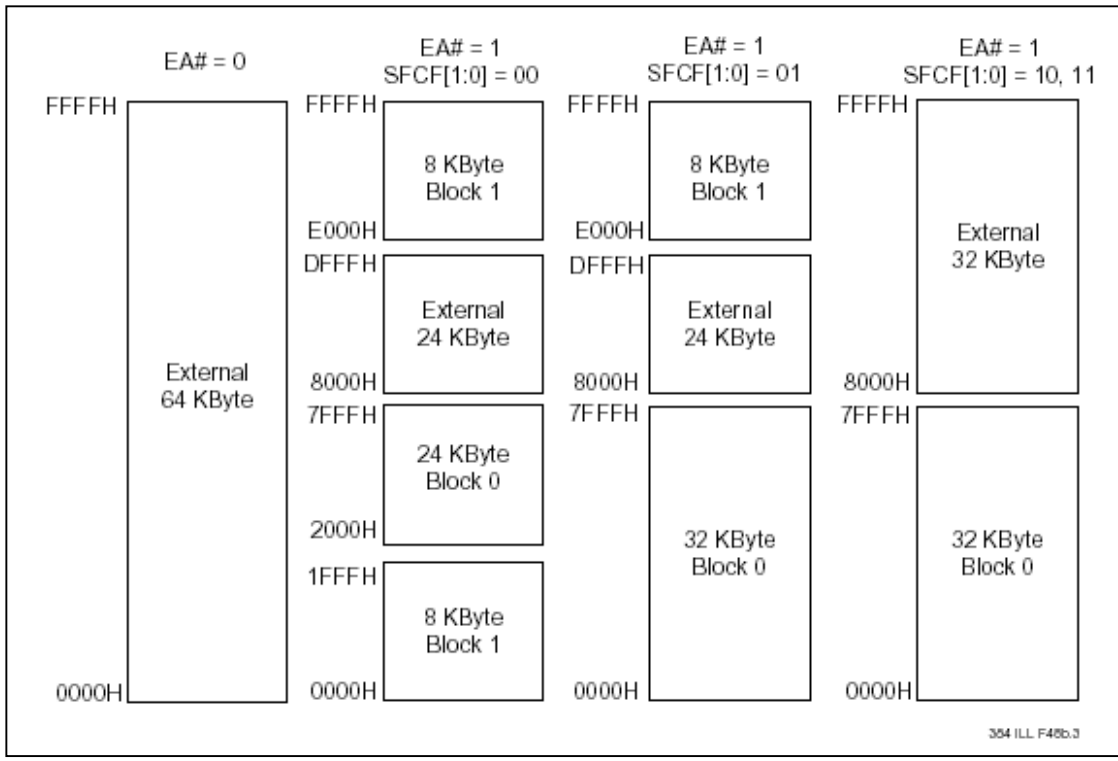


图 9-3-2：SST89E/V58RD 的程序存储器组织(52RD,54RD 与此类同,不同的只是 BLOCK0 的容量， SST89E/V52RD:8K； SST89E/V54RD:16K； SST89E/V58RD:32K.)



单片机允许在程序地址空间的 Block1 或 Block 0 的低 8K 之间切换。SFCF[1：0]控制程序存储块的切换。

表 9-3-1：SFCF 值 程序存储块切换(SST89E/V516RD)

SFCF[1：0]	程序存储块切换
01，10，11	对 PC 来说，Block 1 是禁止的；只有在 IAP 模式，Block 1 从 0000H-1FFFH 可寻址
00	Block 1 覆盖了程序地址空间的低 8K 即 0000H—1FFFH 的地址空间。当 PC 下降到 0000H—1FFFH 时，指令将从 Block 1 中读取而不是从 Block 0 读取。在 0000H—1FFFH 外，Block 0 可用。Block 1 的 0000H—1FFFH 在 IAP 模式下可寻址。

表 9-3-2：SFCF 值-程序存储块切换(SST89E/V52RD,54RD,58RD)

SFCF[1：0]	程序存储块切换
10，11	对 PC 来说，Block 1 是禁止的；在 IAP 模式下，Block 1 从 E000H-FFFFH 可寻址
01	对 PC 来说，Block 0 和 Block 1 都是允许的；Block 0 覆盖了 0000H—7FFFH，Block 1 覆盖了 E000H—FFFFH 的地址空间。
00	Block 1 覆盖了程序地址空间的低 8K 即 0000H—1FFFH 的地址空间。当 PC 下降到 0000H—1FFFH 时，指令将从 Block 1 中读取而不是从 Block 0 中读取。在 0000H—1FFFH 外，Block 0 允许。Block 0 的 0000H—1FFFH 在 IAP 模式下可寻址。

1.3.2.1 程序存储块切换的初始设置

程序存储块切换是在复位后根据 SC0 位的状态来初始化的。SC0 是根据外部主模式命令或 IAP 命令来编程的。可参考表 9-4-2 和 9-4-6。

一旦退出复位，SFCF[0]能根据设计需要通过编程动态改变。改变 SFCF[0]将不会改变 SC0。

在动态改变 SFCF[0]时，必须小心谨慎。因为这将导致不同的物理存储器被映射到逻辑程序地址空间。使用者必须避免在空间 0000H—1FFFH 之间执行块切换命令。

1.3.3 数据存储器

1024*8 片内 RAM,片外寻址 64K 的外部数据存储器。

单片机有四组片内数据存储器：

1. RAM 的低 128 字节（00H—7FH）通过直接和间接寻址访问。
2. RAM 的高 128 字节（80H—FFH）只能通过直接寻址访问。
3. 特殊功能寄存器（SFRS，80H—FFH）只能通过直接寻址访问。
4. 768 字节（00H—2FFH）的内部扩展 RAM 通过清零 EXTRAM 和外部传送指令（MOVX）来实现间接寻址（参考 15 页“辅助寄存器（AUXR）的有关描述）。

表 9-3-3 ：不同复位条件的 SFCF 值

SC1[1]	SC0	在下列模式下 SFCF[1：0]的状态		
		上电或外部复位	WDT 或 Brown-dowm 复位	软件复位
1	1	00	x0	10
1	0	01	x1	11
0	1	10	10	10
0	0	11	11	11

Note1:SC1 只用于 SST89E58RD 和 SST89V58RD。

1.3.4 双数据指针

两个 16 位数据指针。在 AUXR1 中的 DPTR 的选择位（DPS）决定了哪个数据指针是可访问的。当 DPS=0，选择 DPTR0；DPS=1，选择 DPTR1。两个数据指针之间的快速切换可对 AUXR1 执行 INC 指令来实现。

双 DPTR 数据指针给查表和寻址带来很大方便。下面用两个指针将外部数据存储器中以地址 0000H 开始的 64 字节数据送到以 1000H 开始的单元中。

```
        AUXR1       DATA    0A2H
        ; AUXR1
        ; ———————————————————————————————
        ; |  —  |  —  |  —  |  —  |GF2  |   0   |  —  |DPS  |
        ; ———————————————————————————————
        MOV         R1,#64
        INC         AUXR1                   ; 选用数据指针 0
        MOV         DPTR,#0000H
        INC         AUXR1                   ; 切换到数据指针 1
        MOV         DPTR,#1000H
CONVEY:
        INC         AUXR1                   ; 切换到数据指针 0
        MOVX        @DPTR,A
        INC         AUXR1                   ; 切换到数据指针 1
        MOVX        A,@DPTR
        DJNZ        R1,CONVEY
```

1.3.5 特殊功能寄存器

Flash Flex51 单片机系列的大部分特性都是通过如图 9-3-4 中所示的特殊功能寄存器的相应位来控制的。每个 SFR 的单独描述和复位值在表 9-3-5 和 9-3-9 中可见。

表 9-3-4： Flash Flex51 SFR 存储器映射

TABLE 3-4: FLASHFLEX51 SFR MEMORY MAP

	8 BYTES								
F8H	IP1[1]	CH	CCAP0H	CCAP1H	CCAP2H	CCAP3H	CCAP4H		FFH
F0H	B[1]							IP1H	F7H
E8H	IEA[1]	CL	CCAP0L	CCAP1L	CCAP2L	CCAP3L	CCAP4L		EFH
E0H	ACC[1]								E7H
D8H	CCON[1]	CMOD	CCAPM0	CCAPM1	CCAPM2	CCAPM3	CCAPM4		DFH
D0H	PSW[1]					SPCR			D7H
C8H	T2CON[1]	T2MOD	RCAP2L	RCAP2H	TL2	TH2			CFH
C0H	WDTC[1]								C7H
B8H	IP[1]	SADEN							BFH
B0H	P3[1]	SFCF	SFCM	SFAL	SFAH	SFDT	SFST	IPH	B7H
A8H	IE[1]	SADDR	SPSR				XICON		AFH
A0H	P2[1]		AUXR1			P4			A7H
98H	SCON[1]	SBUF							9FH
90H	P1[1]								97H
88H	TCON[1]	TMOD	TL0	TL1	TH0	TH1	AUXR		8FH
80H	P0[1]	SP	DPL	DPH		WDTD	SPDR	PCON	87H

T3-4.0 1255

1. Bit addressable SFRs

表 9-3-5： 与 CPU 相关的特殊功能寄存器(SFR)

符号	描述	地址	位地址，符号，端口功能 MSB LSB								复位值
ACC[1]	累加器	E0H	ACC[7：0]								00H
B[1]	B 寄存器	F0H	B[7：0]								00H
PSW[1]	程序状态字	D0H	CY	AC	F0	RS1	RS0	OV	F1	P	00H
			进位		标志 0			溢出	标志 1	(半进位)	
SP	堆栈指针	81H	SP[7：0]								07H
DPL	指针低 8 位	82H	DPL[7：0]								00H
DPH	指针高 8 位	83H	DPH[7：0]								00H
IE[1]	中断允许	A8H	EA	EC	ET2	ES	ET1	EX1	ET0	EX0	40h
			总允许	PCA	T2	UART/SPI	T1	外部 1	T0	外部 0	
IEA[1]	中断允许 A	E8H	-	-	-	-	EBO	-	-	-	xxxx0xxxb
							低压				
IP	中断优先级寄存器	B8H	-	PPC	PT2	PS	PT1	PX1	PT0	PX0	xx000000b
				PCA	T2	UART/SPI	T1	外部 1	T0	外部 0	
IPH	高中断优先级寄存器	B7H	-	PPCH	PT2H	PSH	PT1H	PX1H	PT0H	PX0H	xx000000b
IP1	中断优先级寄存器 A	F8H	1	-	-		PBO	PX3	PX2	1	1xx10001b
							低压	外部 3	外部 2		
IP1H	高中断优先级寄存器 A	F7H	1	-	-		PBOH	PX3H	PX2H	1	1xx10001b
							低压	外部 3	外部 2		
PCON	电源控制	87H	SMOD1	SMOD0	BOF	POF	GF1	GFO	PD	IDL	00010000b
			波特率加倍	1 选择 FE 0: 选择 SM0	低压复位标志	上电复位标志	通用标志位 1	通用标志位 0	激活掉电模式	激活空闲模式	
AUXR	辅助寄存器	8EH			-		-	-	EXTRAM	AO	xxxxxx00b
									外部 RAM	禁止 ALE	
AUXR1	辅助寄存器 1	A2H			-		GF2	0	-	DPS	xxxx00x0b
							标志 2			DPTR1	
XICON	外部中断控制寄存器	AEH	0	EX3	IE3	IT3	0	EX2	IE2	IT2	00H
				外部 3 允许	外 3 标志	外 3 下降沿触发		外部 2 允许	外 2 标志	外 2 下降沿触发	

TCON[1]	定时/计数器控制	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00H
			T1 溢出标志	T1 启动	T0 溢出标志	T0 启动	外部 1 中断标志	外部 1 下降沿触发	外部 0 中断标志	外部 0 下降沿触发	

表：9-3-6 ：对 FLASH 存储器编程用特殊功能寄存器(SFR)

符号	描述	地址	位地址，符号，端口功能 MSB LSB								复位值
SFST	FLASH 状态	B6H	SB1_i 加密位 1	SB2_i 加密位 2	SB3_i 加密位 3	-	EDC_i 倍速标志	FLASH-BUSY IAP 忙标志	-	-	xxxxx0xxb
SFCF	FLASH 配置	B1H	-	IAPEN IAP 允许	-	-	-	-	SWR 软件复位	BSEL 块选择	x0xxxxxxb
SFCM	FLASH 命令	B2H	FIE 中断允许	FCM IAP 命令字							00H
SFDT	FLASH 数据	B5H	FLASH 数据寄存器								00H
SFAL	FLASH 低 8 位地址	B3H	FLASH 低 8 位地址:A7-A0（SFAL）								00H
SFAH	FLASH 高 8 位地址	B4H	FLASH 高 8 位地址:A15-A0（SFAH）								00H

表 9-3-7：看门狗定时器用特殊功能寄存器

符号	描述	地址	位地址，符号，或转换口作用 MSB……..LSB								复位值
WDTC[1]	看门狗定时器控制	C0H	-	-	-	WDOUT 输出到 RST 管脚	WDRE 复位允许	WDTS 复位标志	WDT 喂狗刷新	SWDT 启动 WDT	xxx00x00b
WDTD	看门狗定时器数据/加载	85H	看门狗的定时数据								00H

表 9-3-8 ：定时/计数器

符号	描述	置接地址	位地，符号，端口功能 MSB LSB								复位值
TMOD	定时/计数器模式控制	89H	定时器 1				定时器 0				00H
			GATE	C/T#	M1	M0	GATE	C/T#	M1	M0	
			受 INT1 控制	计数/定时	模式选择		受 INT0 控制	计数/定时	模式选择		
TCON[1]	定时/计数器控制	88H	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0	00H
			T1 溢出标志	T1 启动	T0 溢出标志	T0 启动	外部 1 中断标志	外部 1 下降沿触发	外部 0 中断标志	外部 0 下降沿触发	
TH0	定时器 0	8CH	TH0[7：0] T0 定时常数高 8 位								00H
TL0	定时器 0	8AH	TL0[7：0] T0 定时常数低 8 位								00H
TH1	定时器 1	8DH	TH1[7：0] T1 定时常数高 8 位								00H
TL1	定时器 1	8BH	TL1[7：0] T1 定时常数低 8 位								00H
T2CON[1]	定时/计数器 2 控制	C8H	TF2	EXF2	RCLK	TCLK	EXEN2	TR2	C/T2#	CP/RL2#	00H
			T2 溢出标志	T2 外部标志	接收时钟标志	发送时钟标志	外部使能	启动	外部计数	捕获/重装	
T2MOD#	定时/计数器 2 模式控制	C9H	-	-	-	-	-	-	T2OE	DCEN	xxxxxx00b
									输出使能	递减使能	
TH2	定时器 2	CDH	TH2[7：0] T2 定时常数高 8 位								00H
TL2	定时器 2	CCH	TL2[7：0] T2 定时常数低 8 位								00H
RCAP2H	T2 捕捉/重装寄存器	CBH	RCAP2H[7：0] 高 8 位								00H
RCAP2L	T2 捕捉/重装寄存器	CAH	RCAP2L[7：0] 低 8 位								00H

1.可位寻址的 SFRS.

表 9-3-9：接口用特殊功能寄存器(SFR)

符号	描述	直接地址	位地，符号，端口功能 MSB LSB								复位值
SBUF	串行数据缓冲器	99H	SBUF[7：0]								结束
SCON[1]	串行端口控制	98H	SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI	00H
			FE:帧错误标志	模式选择 1	模式选择 2	接收允许	发送的第 9 位	接收的第 9 位	发送中断标志	接收中断标志	
SADDR	从机地址/串口地址寄存器	A9H	SADDR#[7：0] 与 SADEN”与”得出广播响应的地址								00H
SADEN	从机地址标志/串口地址使能	B9H	SADEN#[7：0]								00H
SPCR	SPI 控制寄存器	D5H	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	04H
			中断使能	SPI 使能	低位先传	主机模式	空闲 SCK 高	时钟后沿触发	主机的分频选择 0-3 4,16,64,128		
SPSR	SPI 状态寄存器	AAH	SPIF	WCOL							00H
			SPI 中断标志	写冲突标志							
SPDR	SPI 数据寄存器	86H	SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0	00H
P0[1]	端口 0	80H	P0[7：0]								FFH
P1[1]	端口 1	90H	CEX4/SPICLK	CEX3/MISO	CEX2/MOSI	CEX1/SS	CEX0	ECI	T2EX	T2	FFH
			比较/捕捉输入输出管脚 4 /SPI 时钟	比较/捕捉输入输出管脚 3	比较/捕捉输入输出管脚 2	比较/捕捉输入输出管脚 1 片选	比较/捕捉输入输出管脚 0	PCA 外部时钟输入	捕获/重装触发和方向控制	T2 的计数输入/时钟输出	
P2[1]	端口 2	A0H	P2[7：0]								FFH
P3[1]	端口 3	B0H	RD#	WR#	T1	T0	INT1#	INT0#	TXD	RXD	FFH
P4	端口 4	A5H	1	1	1	1	P4.3	P4.2	P4.1	P4.0	FFH

表 9-3-10 ：PCA 特殊功能寄存器

符号	描述	地址	位地址，符号，或可选择口或能 MSB LSB								复位值
CH CL	PCA 定时/计数器	F9H E9H	CH[7：0] PCA 定时器高 8 位 CL[7：0] PCA 定时器低 8 位								00H
CCON	PCA 定时器/计数器控制寄存器	D8H	CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0	00H
			计数器溢出标志	启动 PCA 计数器		PCA4 中断标志	PCA3 中断标志	PCA2 中断标志	PCA1 中断标志	PCA0 中断标志	
CMOD	PCA 定时器/计数器模式寄存器	D9H	CIDL	WDTE	-	-	-	CPS1	CPS0	ECF	00x00000b
			空闲禁能	PCA 的 WDG 使能				计数源选择 0-F/12; 1-F/4 2-T0; 3-P1.2		使能计数溢出中断	
CCAP0H	PCA 模块 0 比较/捕捉寄存器	FAH	CCAP0H[7：0]								00xxx000b
CCAP0L		EAH	CCAP0L[7：0]								00H
CCAP1H	PCA 模块 1 比较/捕捉寄存器	FBH	CCAP1H[7：0]								00H
CCAP1L		EBH	CCAP1L[7：0]								00H
CCAP2H	PCA 模块 2 比较/捕捉寄存器	FCH	CCAP2H[7：0]								00H
CCAP2L		ECH	CCAP2L[7：0]								00H
CCAP3H	PCA 模块 3 比较/捕捉寄存器	FDH	CCAP3H[7：0]								00H
CCAP3L		EDH	CCAP3L[7：0]								00H
CCAP4H	PCA 模块 4 比较/捕捉寄存器	FEH	CCAP4H[7：0]								00H
CCAP4L		EEH	CCAP4L[7：0]								00H
CCAPM0	PCA 比较/捕捉模块模式寄存器	DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000b
				使能比较器	上升沿捕获	下降沿捕获	匹配使能	匹配翻转 CEX 管脚	使能 PWM 管脚输出	使能 CCF 中断	
CCAPM1		DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000b
CCAPM2		DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x0000000b
CCAPM3		DDH	-	ECOM3	CAPP3	CAPN3	MAT3	TOG3	PWM3	ECCF3	x0000000b
CCAPM4		DEH	-	ECOM4	CAPP4	CAPN4	MAT4	TOG4	PWM4	ECCF4	x0000000b

SuperFlash 状态寄存器（只读寄存器）SuperFlash Status Register（SFST）

7	6	5	4	3	2	1	0
SECD1	SECD2	SECD3	-	-	FLASH_BUSY	-	-

地址 0B6H 复位值 xxxxx0xxb

符号	功能	符号	功能	符号	功能
SECD1	安全位 1	SECD2	安全位 2	SECD3	安全位 3

FLASH-BUSY FLASH 操作完成检测位

1: 芯片忙

2: 芯片已完成 IAP 命令

SuperFlash 配置寄存器 SuperFlash Configuration Register（SFCF）

地址 0B1H 复位值 xxxxx0xxb

7	6	5	4	3	2	1	0
	IAPEN					SWR	BSEL

符号	功能
IAPEN	IAP 操作允许 0：IAP 命令禁止 1：IAP 命令允许
SWR	软件复位(参见 “1.2 软件复位”)
BSEL	程序存储块切换位(参见 9-3-1 和 9-3-2 中的有关描述)

SuperFlash 命令寄存器(SFCM)

地址 0B2H 复位值 00000000b

7	6	5	4	3	2	1	0
FIE	FCM6	FCM5	FCM4	FCM3	FCM2	FCM1	FCM0

符号	功能
FIE	FLASH 中断允许位 0:INT1#不允许再被指定为 IAP 中断 1:INT1#被指定为 IAP 操作完成标志，外部 INT1# 中断忽略
FCM[6:0]	FLASH 操作命令 000-1011B 扇区-擦除（Sector-Erase） 000-1101B 块-擦除(Block-Erase) 000-1100B 字节-校验(Byte-Verify) 000-1110B 字节-编程(Byte-Program) 000-1111B 编程-SB1 位(Prog-SB1) 000-0011B 编程-SB2 位(Prog-SB2)

000-0101B 编程-SB3 位(Prog-SB3)

000-1001B 编程-SC0 位(Prog-SC0)

其它的组合没有实现，保留以后使用。

注 1．字节校验有一个单机器周期反应时间，不管 FIE 如何，它不产生 INT1#中断.

SuperFlash 数据寄存器（SFDT）

地址 0B5H 复位值 00000000b

7	6	5	4	3	2	1	0
SuperFlash 数据寄存器							

符号 功能

SFDF 与 FLASH 存储块接口的邮箱寄存器（数据寄存器）

SuperFlash 低 8 位地址寄存器（SFAL）

地址:0B3H 复位值 00000000b

7	6	5	4	3	2	1	0
SuperFlash 低 8 位地址寄存器							

符号 功能

SFAL 用来与 FLASH 程序存储器接口的邮箱寄存器（低 8 位地址寄存器）

SuperFlash 高 8 位地址寄存器（SFAH）

地址 0B4H 复位值 00000000b

7	6	5	4	3	2	1	0
SuperFlash 高 8 位地址寄存器							

符号 功能

SFAH 用来与 FLASH 程序存储器接口的邮箱寄存器(高 8 位地址寄存器)

器件支持 8 中断源，4 中断优先级的中断结构。表 43 总结了支持中断的查询序列。注意：SPI 串行接口和 UART 共用同一个中断向量（见图 27)。

表 43 中断查询序列

描述	中断标志	向量地址	中断使能	中断优先级	服务优先级	掉电唤醒
外部中断 0	IE0	0003H	EX0	PX0/H 1	（最高）	能
掉电	-	004BH	EBO	PBO/H	2	不能
T0	TF0	000BH ET0		PT0/H	3 不	能
外部中断 1 /IAP	IE1 /SF	0013H	EX1	PX1/H	4	能
T1	TF1	001BH ET1		PT1/H	5 不	能
PCA	CF/CCFn	0033H	EC	PPCH	6	不能
UART/SPI	TI/RI/SPIF	0023H	ES	PS/H	7	不能
T2	TF2,EXF2	002BH	ET2	PT2/H	8	不能

中断允许寄存器 Interrupt Enable（IE）

地址 A8H 复位值 00H

7	6	5	4	3	2	1	0
EA	EC	ET2	ES	ET1	EX1	ET0	EX0

符号	功能
EA	中断允许 0=CPU 屏蔽所有的中断申请 1=CPU 开放所有的中断申请
EC	PCA 中断允许
ET2	定时器 2 中断允许
ES	串行口中断允许
ET1	定时器 1 中断允许
EX1	外部中断 1 允许
ET0	定时器 0 中断允许
EX0	外部中断 0 允许

中断允许寄存器 A Interrupt Enable A （IEA）

地址 E8H 复位值 xxxx0xxxb

7	6	5	4	3	2	1	2
-	-	-	-	EBO	-	-	-

符号	功能
EBO	Brown-out 中断允许 1=中断允许 0=中断禁止

中断优先级寄存器 Interrupt Priority（IP）

地址 B8H 复位值 xx000000b

7	6	5	4	3	2	1	0
-	PPC	PT2	PS	PT1	PX1	PT0	PX0

符号	功能
PPC	PCA 中断优先位
PT2	定时器 2 中断优先位
PS	串行口中断优先位
PT1	定时器 1 中断优先位
PX1	外部中断 1 优先位
PT0	定时器 0 中断优先位
PX0	外部中断 0 优先位

中断优先级高 Interrupt Priority High（IPH）

地址 B7H 复位值 xx000000b

7	6	5	4	3	2	1	0
-	-	PT2H	PSH	PT1H	PX1H	PT0H	PX0H

符号	功能
PT2 H	定时器 2 中断优先位高
PSH	串行口中断优先位高
PT1H	定时器 1 中断优先位高
PX1H	外部中断 1 优先位高
PT0H	定时器 0 中断优先位高
PX0H	外部中断 0 优先位高

中断优先 A 寄存器 Interrupt Priority A（ IPA）

地址 F8H 复位值 xxxx0xxxb

7	6	5	4	3	2	1	0
-	-	-	-	PBO	-	-	-

符号	功能
PBO	(Brown-out) 低电压检测中断优先位

高中断优先寄存器 A Interrupt Priority High（ IPAH）

地址 F7H 复位值 xxxx0xxxb

7	6	5	4	3	2	1	0
-	-	-	-	PBOH	-	-	-

符号	功能
PBOH	Brown-out 中断优先位高

辅助寄存器 Auxiliary Registe（AUXR）

地址 8EH 复位值 xxxxxx00b

7	6	5	4	3	2	1	0
-	-	-	-	-	-	EXTRAM	AO

符号 功能

EXTRAM 0：内部扩展 RAM 存取，具体可参考 12 页的“数据存储器”。
1：外部数据存储器存取。

AO 0：ALE，正常方式
1：ALE 正常情况下被关闭。只有在执行 MOVX 和 MOVC 指令才会工作，这将减少 EMI。

辅助寄存器 1 Auxiliary Register 1（AUXR1）

地址 A2H 复位值 xxxx00x0b

7	6	5	4	3	2	1	0
-	-	-	-	GF2	0	-	DPS

符号 功能

GF2 用户自定义通用标志位

DPS DPTR 寄存器选择位
0：DPTR0 被选择
1：DPTR1 被选择

看门狗定时器控制寄存器 Watchdog Timer Control Register（WDTC）

地址: 0C0H 复位值 xxx00x00b

7	6	5	4	3	2	1	0
-	-	-	WDOUT	WDRE	WDTS	WDT	SWDT

符号 功能

WDOUT 看门狗输出允许

0：看门狗复位将不从 RESET 输出

1：如果看门狗复位被 WDRE 允许，将复位 RESET 引脚 32 个时钟。

WDRE 看门狗定时器复位允许

0：禁止复位

1：允许复位

WDTS 看门狗定时器复位标志

0：外部硬件复位清零标志位

标志位也可通过写入“1”来清除,

标志位将会保持，如果由于看门狗定时器溢出而引起芯片复位。

1：硬件置位，当看门狗定时器溢出时。

WDT 看门狗定时器刷新

0：当刷新完成时，硬件复位此位。

1：软件置位此位强迫看门狗定时器刷新。

SWDT 启动看门狗定时器

0：停止 WDT

1：启动 WDT

看门狗定时器数据寄存器（WDTD）

地址 085H 复位值 00000000b

7	6	5	4	3	2	1	0
看门狗定时器 数据							

符号 功能

WDTD 初始化/重载看门狗定时器内的值。当 WDT 置位后新值才有效。

溢出时间的计算公式:

溢出时间=(255-WDTD) * 344064 * 1/fclk(晶振频率)

PCA 定时器/计数器控制寄存器（CCON）

地址 D8H 复位值 00x00000b

7	6	5	4	3	2	1	0
CF	CR	-	CCF4	CCF3	CCF2	CCF1	CCF0

符号 功能

CF PCA 定时器/计数器溢出标志

当 PCA 定时/计数器溢出时由硬件置位.如果 CMOD 中的中断允许位 ECF=1,将产生一个中断.CF 能被硬件和软件置位但只能通过软件清零.

CR PCA 定时器/计数器运行控制位

当匹配或锁存发生时由硬件置位.如果相应的 CCAPMX 寄存器中的中断允许位 ECCFX 被置 1 时将产生一个 PCA 中断请求.必须通过软件清零.

CCF[4:0] PCA 模块比较/捕捉标志:

当匹配或锁存发生时由硬件置位. 如果相应的 CCAPMX 寄存器中的中断允许位 ECCFX 被置 1 时将产生一个 PCA 中断请求.必须通过软件清零.

PCA 定时器/计数器模式寄存器(CMOD)

地址 D9H 复位值 00xxx000b

7	6	5	4	3	2	1	0
CIDL	WDTE	-	-	-	CPS1	CPS0	ECF

符号 功能

CIDL PCA 定时/计数器 Idle 控制

0:允许 PCA 定时/计数器在 Idle 模式下运行

1:在 Idle 模式下禁止 PCA 定时/计数器运行

WDTE 看门狗定时器允许

0：禁止 PCA 看门狗定时器输出

1：允许 PCA 看门狗定时器在 PCA 模式 4 下输出

CPS1，CPS0 看门狗定时器/计数器输入选择

CPS1	CPS0	
0	0	fosc/12(双倍速模式: fosc/6)
0	1	fosc/4(双倍速模式: fosc/2)
1	0	定时器 0 溢出
1	1	ECI 引脚上的外部时钟（最大频率=fosc/8(双倍速模式: fosc/4)）

ECF PCA 定时/计数器中断允许

0：禁止 CCON 寄存器中的 CF 位

1：允许 CCON 寄存器有 CF 位来产生一中断请求

表 9-3-11：PCA 模块模式

ECOMy[1]	CAPPy[1]	CAPNy[1]	MATy[1]	TOGy[1]	PWNy[1]	ECCFy[1]	Module Code
0	0	0	0	0	0	0	No Operation
-	1	0	0	0	0	-	16-bit capture on positive-edge trigger at CEX[4:0]
-	0	1	0	0	0	-	16-bit capture on negative-edge trigger at CEX[4:0]
-	1	1	0	0	0	-	16-bit capture on positive-/negative-edge trigger at CEX[4:0]
1	0	0	1	0	0	-	Compare: software timer
1	0	0	1	1	0	-	Compare: high-speed output
1	0	0	0	0	1	0	Compare: 8-bit PWM
1	0	0	1	-	0	-	Compare: PCA WDT (CCAPM4 only)[2]

T3-11.0 555

1. y = 0, 1, 2, 3, 4
2. For PCA WDT mode, also set the WDTE bit in the CMOD register to enable the reset output signal

PCA 比较/捕捉模块模式寄存器（CCAPM[4：0]）

Location	7	6	5	4	3	2	1	0	Reset Value
DAH	-	ECOM0	CAPP0	CAPN0	MAT0	TOG0	PWM0	ECCF0	x0000000b
DBH	-	ECOM1	CAPP1	CAPN1	MAT1	TOG1	PWM1	ECCF1	x0000000b
DCH	-	ECOM2	CAPP2	CAPN2	MAT2	TOG2	PWM2	ECCF2	x0000000b
DDH	-	ECOM3	CAPP3	CAPN3	MAT3	TOG3	PWM3	ECCF3	x0000000b
DEH	-	ECOM4	CAPP4	CAPN4	MAT4	TOG4	PWM4	ECCF4	x0000000b

符号 功能

ECOM[4：0] 比较模式

0：禁止模块比较器功能

1：允许模块比较器功能。比较器用来用作软件定时器，高速输入，脉冲宽度调节，看门狗定时器模式。

CAPP[4：0] 0：通过加在 CEX[4：0]上的上升沿禁止锁存功能同时锁存器触发。

1：通过加在 CEX[4：0]上的上升沿允许锁存功能同时锁存器触发。

CAPN[4：0] 0：通过加在 CEX[4：0]上的下降沿禁止锁存功能同时锁存器触发。

1：通过加在 CEX[4：0]上的下降沿允许锁存功能同时锁存器触发。

MAT[4：0] 匹配：置位 ECOM[4：0]和 MAT[4：0]用作软件定时器模式

0：禁止软件定时器模式

1：PCA 定时器/计数器的一个匹配，比较/捕捉寄存器置位 CCON 寄存器的 CCF[4：0]，标志一个中断。

TOG[4：0] 触发：

置位 ECOM[4：0]、MAT[4：0]、TOG[4：0]来实现高速输入模式

0：禁止触发功能

1：PCA 定时器/计数器的一个匹配，比较/捕捉寄存器触 CEX[4：0]引脚。

PWM[4：0] 脉冲宽度调节模式

0：禁止脉冲宽度调节模式

1：设置模块为一个 8 位的脉冲宽度调节器，输出波形到 CEX[4：0]引脚

ECCF[4：0] 允许 CCF[4：0]中断

0：禁止 CCON 寄存器的比较/锁存标志 CCF[4：0]产生中断请求

1：允许 CCON 寄存器的比较/锁存标志 CCF[4：0]产生中断请求

SPI Control Register （SPCR）——SPI 控制寄存器

地址 D5H 复位值 00000100B

7	6	5	4	3	2	1	0
SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0

符号 功能

SPIE 如果 SPIE 和 ES 都被置为“1”，SPI 中断被允许。

SPE SPI 允许位。

0：禁止 SPI

1：允许 SPI（P1[4] ，P1[5]，P1[6]，P1[7]为 SS#、MOSI、MISO、SCK）

DORD 数据传输顺序。

0：从 MSB（最高位）开始传送

1：从 LSB（最低位）开始传送

MSTR 主/从选择

0：选择从模式; 1：选择主模式

CPOL 时钟极性

0：在 Idle 时，SCK 处于低（Active High）

1：在 Idle 时，SCK 处于高（Active Low）

CPHA 时钟相位控制位

0：移位触发在时钟上升沿

1：移位触发在时钟下降沿

SPR1.SPR0: SPI 时钟频率选择位。

这两位控制主机的 SCK 频率，SPR1 和 SPR0 对从机没有影响。

SCK 和振荡器频率(f_{OSC})的关系表示如下：

SPR1	SPR0	SCK= f_{OSC} 除以下面的值
0	0	4
0	1	16
1	0	64
1	1	128

SPI Status Register（SPSR）——SPI 状态寄存器

地址 AAH 复位值 00xxxxxb

7	6	5	4	3	2	1	0
SPIF	WCOL	-	-	-	-	-	-

符号	功能
SPIF	置位 “1”，完成数据传送后，如果 SPIE=1，ES=1，将会产生中断。要清除、读 SPSR，存取 SPDR。
WCOL	置位 “1”，如果 SPI 数据寄存器在数据传送过程中被写入了数据, 要清除，读 SPSR，存取 SPDR。

SPI Dtat Register（SPDR）——SPI 数据寄存器

地址 86H 复位值 00H

7	6	5	4	3	2	1	0
SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0

Power Control Register（PCON）——电源控制寄存器

地址 87H 复位值 00010000b

7	6	5	4	3	2	1	0
SMOD1	SMOD0	BOF	POF	GF1	GF0	PD	IDL

符号	功能
SMOD1	波特率倍增控制位。如果 SMOD1=1，定时器 1 被用于产生波特率。
SMOD0	FE/SM0 选择位 0： SCON[7]=SM0 1： SCON[7]=FE。
BOF	Brown-out 检测状态位，这个位不会被任何其它的复位影响。BOF 可以通过软件清零。上电复位也将清零 BOF 位。 0：无 Brown_out 1：Brown_out 产生
POF	上电复位状态位，这个位不会被任何其它的复位影响。POF 可以通过软件清零。上电复位也将清零 POF 位。 0：无上电复位; 1：加电复位发生
GF1	通用标志位
GF0	通用标志位
PD	Power_dowm 位 0：Power_dowm 不被激活; 1：Power_dowm 激活

IDL Idle 模式位

0：Idle 模式禁止; 1：Idle 模式激活

Serial Port Control Register（SCON）——串行口控制寄存器

地址 98H 复位值 00000000b

7	6	5	4	3	2	1	0
SM0/FE	SM1	SM2	REN	TB8	RB8	TI	RI

符号 功能

FE SMOD0=1: SCON[7]=FE

0：无帧错误。

1：帧错误。当检测到无效的停止位时，接收方置位此位。这个位需要通过软件清零。

SM0 SMOD0=0：SCON[7]=SM0

串行口模式位 0

SM1 串行口模式位 1

SM0	SM1	MODE	描述	波特率
0	0	0	移位寄存器	f_{OSC}/6(6 clock mode)或 f_{OSC} /12(12clock mode)
0	1	1	8 位 UART	可变
1	0	2	9 位 UART	f_{OSC} /32 或 f_{OSC} /16(6 clock mode)或 f_{OSC} /32 或 f_{OSC} /64(12 clock mode)
1	1	3	9 位 UART	可变

1. f_{OSC}=振荡器频率。

SM2 允许自动地址识别，在模式 2 或模式 3 下。如果 SM2=1，那么 RI 不会被置位除非接收到第九位数据（RB8）是‘1’时，表明是一个地址，接收到的是给定的或广播地址。在模式 1 下，如果 SM2=1 ，那么 RI 将不被激活，除非接收到一有效的停止位，并且接收到的是给定的或广播地址。在模式 0 下，SM2 应为 0。

REN 允许串行接收控制位

0：禁止接收。 1：允许接收。

TB8 在模式 2 或 3 下，发送的第 9 位数据，可以通过软件置位或清零。

RB8 在模式 2 或 3 下，接收到的第 9 位数据。在模式 1 下，如果 SM2=0，RB8 是接收到的停止位。在模式 0 下，RB8 不用。

TI 发送中断标志位。在模式 0 下，在发送完第 8 位数据时由硬件置位，在其它模式下在停止位的开始被置位，在任何串行通信中，必须通过软件清零。

RI 接收中断标志位。在模式 0 下，在接收完第 8 位数据位时由硬件置位，或在其它模式下在停止位的中间被置位（其它的可查看 SM2 的有关说明）。此位必须通过软件清零。

1.4 FLASH 存储器编程

单片机的内部 FLASH 存储器在下列两种模式下能被编程或擦除：

—外部主模式

—IAP 模式（In-Application Programming Mode）

1.4.1 外部主机编程模式

外部主模式允许用户直接对 FLASH 存储器编程, 不用被编程从机的 CPU。当 RST 的输入一直保持高电平，通过强迫 PSEN#由逻辑高转为逻辑低，单片机这时进入外部主模式。维持 RST=1 和 PSEN#=0 单片机将一直保持在外部主模式。

在外部主模式下，一个 Read-ID 操作是必须的，用来指定器件进入外部主模式状态。在读 ID 指令完成后，其它的指令才允许。在外部主模式下， 内部 FLASH 存储块能通过被外部主机重新配置的 I/O 口引脚来访问，这些外部主机可以是 MCU 编程器、PCB 测试器或 PC 控制开发板。

表 9-4-1：外部主模式命令(SST89E/V516RD)

Operation	RST	PSEN#	PROG#/ ALE	EA#	P3[7]	P3[6]	P2[7]	P2[6]	P0[7:0]	P3[5:4] P2[5:0]	P1[7:0]
Read-ID	V_{IH1}	V_{IL}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	DO	AH	AL
Chip-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}	X	X	X
Block-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IL}	V_{IH}	X	X	X
Sector-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IH}	V_{IH}	X	AH	AL
Byte-Program	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IL}	DI	AH	AL
Byte-Verify (Read)	V_{IH1}	V_{IL}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	DO	AH	AL
Select-Block0	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{II}	V_{II}	V_{IH}	X	55H	X
Select-Block1	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{II}	V_{II}	V_{IH}	X	A5H	X
Prog-SC0	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	5AH	X
Prog-SB1	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	X	X	X
Prog-SB2	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	X	X	X
Prog-SB3	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IL}	V_{IH}	V_{IL}	V_{IH}	X	X	X

注意：符号 ⇓ 表示一个负脉冲,当 PROG#/ALE 输入为低电平时引入指令。上面输入引脚所有其它组合都是无效的，并且可能产生无法预料的情况。

V_{IL}=输入低电压；V_{IH}=输入高电压;V_{IH1}=输入高电压(XTAL,RST);X=不必注意；

AL=地址低 8 位， AH=地址高 8 位;DI=数据输入;DO=数据输出;

表 9-4-2 : 外部主模式命令(SST89E/V52RD,54RD,58RD)

Operation	RST	PSEN#	PROG#/ ALE	EA#	P3[7]	P3[6]	P2[7]	P2[6]	P0[7:0]	P3[5:4] P2[5:0]	P1[7:0]
Read-ID	V_{IH1}	V_{IL}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}	V_{IL}	DO	AH	AL
Chip-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IL}	X	X	X
Block-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IL}	V_{IH}	X	A[15:13]	X
Sector-Erase	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IH}	V_{IH}	X	AH	AL
Byte-Program	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IL}	DI	AH	AL
Byte-Verify (Read)	V_{IH1}	V_{IL}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IL}	V_{IL}	DO	AH	AL
Prog-SC0	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	5AH	X
Prog-SC1	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IL}	V_{IL}	V_{IH}	X	AAH	X
Prog-SB1	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IH}	V_{IH}	V_{IH}	V_{IH}	X	X	X
Prog-SB2	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IL}	V_{IL}	V_{IH}	V_{IH}	X	X	X
Prog-SB3	V_{IH1}	V_{IL}	⇓	V_{IH}	V_{IL}	V_{IH}	V_{IL}	V_{IH}	X	X	X

说明：符号 ⇓ 表示一个负脉冲,当 PROG#/ALE 输入为低电平时引入指令。上面输入引脚所有其它组合都是无效的并且可能产生无法预料的情况。

V_{IL}=输入低电压；V_{IH}=输入高电压；V_{IH1}=输入高电压(XTAL,RST)；X=不必注意；AL=地址低

8 位，AH=地址高 8 位；DI=数据输入；DO=数据输出;；A[15:13]=0xxb（Block 0）

A[15:13]=111b （ Block 1）。

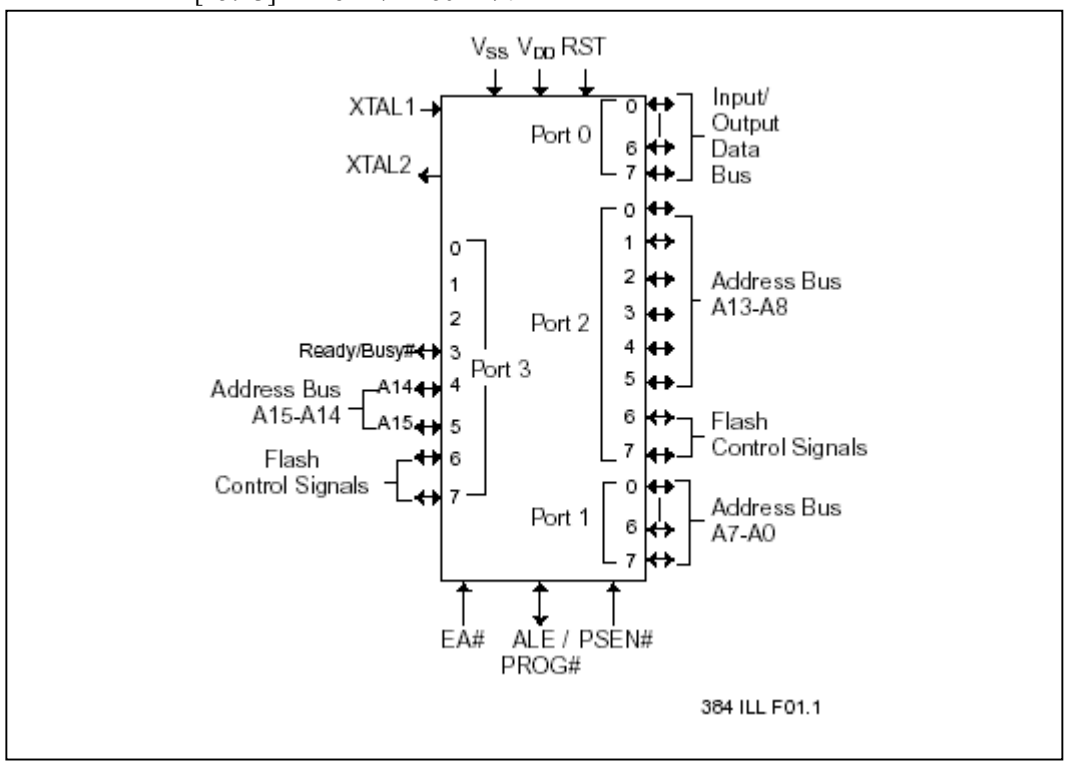


图 9-4-1：外部主模式下 I/O 口的分配

1.4.1.1 产品标识

Read-ID 指令通过访问签名字节来辨别器件和制造商。编程器首先用这些签名字节来选择编程算法。Read-ID 命令被选择通过在 P3[7：6]和 P2[7：6]上的命令码 0H。参见图 4-2 中时序波形

表 9-4-3 签名字节

	地址	数据
生产厂家 ID	30H	BFH
器件 ID		
SST89E52RD	31H	9CH
SST89V52RD	31H	9DH
SST89E54RD	31H	9EH
SST89V54RD	31H	9FH
SST89E58RD	31H	9BH
SST89V58RD	31H	9AH
SST89E516RD	31H	92H
SST89V516RD	31H	93H

1.4.1.2 进入命令

一个进入命令序列必须在外部主机命令被单片机识别之前产生。这防止由于外部噪音或编程器出错而意外触发外部主模式命令。进入命令如下：

1．PSEN#变为低，当 RST 处于高电平时。这将使单片机进入外部主模式，重新设置引脚、启动片上振荡器。

2．发 Read-ID 命令，1mS 后外部主模式命令才能被发出。

在以上序列之后，其它的外部主命令才被允许。在 Read-ID 命令被接收之前，所有接收到的其它外部主模式命令都被忽略。

1.4.1.3 外部主模式命令的详细说明

外部主模式命令有：Read-ID 、Chip-Erase、Block-Erase、Sector Erase、Byte-Program、Byte-Verify、Prog-SB1、Prog-SB2、Prog-SB3、Prog-SC0、Prog-SC1、Select-Block0、Select-Block1。参见表 4-1 和 4-2 中信号逻辑分配，图 4-1 关于 I/O 引脚的分配,表 4-7 时序参数。所有擦除和编程命令的典型时序都是通过片内 FLASH 存储器控制器产生的。PROG#由高到低的变换初始化擦除和编辑命（内部同步）。读命令是异步读，独立于 PROG#信号电平。

下面是外部主模式命令的详细描述：

Select-Block 0 命令允许 Block 0 在外部主模式时可被编程。一旦这个命令被执行，所有后续的外部主模式命令都针对 Block 0。参见图 9-4-3 时序波形图。这个命令只用于 SST89E516RD 和 SST89V516RD。

Select-Block 1 命令允许 Block 1 (8K Block)可被编程。一旦这个命令被执行，所有指向地址范围低于 2000H 的外部主模式命令都将针对 Block 1。Select-Block 1 命令只影响程序存储空间的低 8K 字节。因为如果地址大于或等于 2000H ，Block 0 为默认访问区域。一进入外部主模式，Block1 缺省被选择。参见图 9-4-3 的时序波形图。这个命令只适用于 SST89E516RD 和 SST89V516RD。

Chip-Erase、Block-Erase、和 Sector-Erase 命令用来擦除所有或部分存储器阵列。在存储阵列中被擦除的单元被置为 FFH。在存储器中那些将要被编程的单元在编程前必须已被擦除。

Chip-Erase 命令擦除两个存储块中的所有字节,不管先前的 Select-Block 0 或 Select-Block 1 命令 。Chip-Erase 忽略加密状态并且将擦除加密锁,使器件返回到非加密状态。Chip-Erase 命令也将擦除 SC0 位。一旦 Chip-Erase 命令完成,Block 1 将是被选择的块。参见图 9-4-4 的时序波形图。

Block-Erase 命令擦除被选中存储块中的所有字节。如果加密允许,这个命令将不被执行。擦除哪一存储块依赖于先前执行的是 Select-Block 1 还是 Select-Block 0 命令。参见图 9-4-6 的时序波形图。

Sector-Erase 命令擦除一个扇区内的所有字节。扇区大小为 128 字节。如果加密允许,这个命令将不被执行。参见图 9-4-7 的时序波形图。

Byte-Progrm 命令用来将新数据编程到存储阵列。如果如何加密锁允许,编程将不会发生。参见图 4-8 的时序波形图。

Byte-Verify 命令允许用户去校验单片机是否正确执行了一个擦除或编程命令。如果任何加密锁被允许,这个命令将不被执行。参见图 4-11 的时序波形图。

Prog-SB1、Prog-SB2、Prog-SB3 命令编程加密位，这些位的功能在加密锁那一节和表 8-1 中有说明。一旦被编程，这些位只有通过 Chip-Erase 命令才能被擦除。参见图 4-9 的时序波形图。

Prog-SC 0 命令编程 SC0 位，SC0 位决定了复位后 SFCF[0]的状态。一旦编程，SC0 只能通过 Chip-Erase 命令来恢复为已擦除状态。参见图 4-10 的时序波形图。

Prog-SC 1 命令编程 SC1 位，它用来决定复位后 SFCF[1]的状态。一旦编程，SC1 只能通过 Chip-Erase 命令来恢复为已擦除状态。参见图 4-10 的时序波形图。Prog-SC1 只适用于 SST89E58RD 和 SSTV554。

1.4.1.4 外部主模式时钟源

在外部主模式，一个内部振荡器为单片机提供时钟。在单片机进入外部主模式时，片内振荡器启动，也就是当 PSEN#转为低,RST 为高电平。在外部主模式，CPU 核保持在复位状态，一旦从外部主模式退出，内部振荡器被关闭。

1.4.1.5 FLASH 操作状态检测（通过外部主机握手信号）

单片机提供两种方式给外部主机检测 FLASH 存储器操作的完成，以优化编程或擦除时间。FLASH 存储器操作周期的结束能通过如下被检测：

1. 监测 READY/BUSY#位--P3[3]。
2. 监测 Data# 查询位-- P0[3]。

Ready/Busy# （P3[3]）

FLASH 存储器编程的过程可通过 Ready/Busy#的输出信号来检测。在 FLASH 存储器操作期间，在 ALE/PROG#转为低之后一段时间，P3[3]被驱动为低，指明 Flash 控制单元（FCU）处于 Busy#状态。当 FLASH 编程操作完成时，P3[3]被驱动为高，来指明处于等待状态。

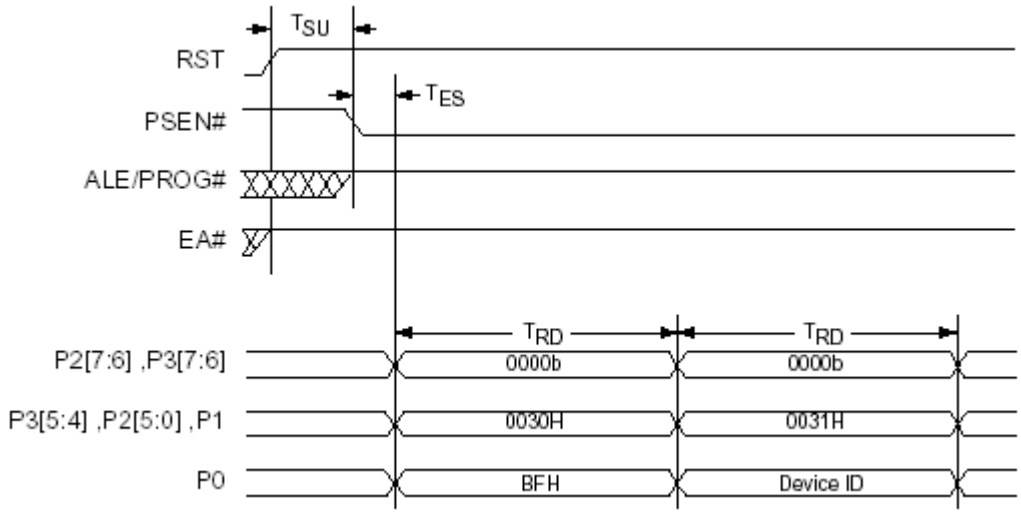
数据查询（P0[3]）在一个编程操作期间，当器件忙时，任何读的企图（Byte-Verify）将接收到最后装载在 P0[3]上字节的数据的完成（逻辑低，也就是‘0’-erase）。 在编程操作期间，Byte-Verify 命令读的是装入的最后一字节，不是指定地址的数据。

1.4.1.6 逐步执行指令来实现外部主模式命令

把数据编程进存储器阵列，提供的电源（V_{DD}）到 V_{DD} 和 RST 引脚，并执行如下步骤：

1. 保持 RST 为高，将 PSEN#由逻辑高置为低，根据相应的时序图来依次完成。
2. 将 EA#拉升为高电平（V_{IH}）。
3. 发出 Read-ID 命令允许外部主模式。
4. 校验将要编程的存储块或扇区处于已擦除状态，FFH。如果没被擦除，用相应的擦除命令擦除它们。
5. 选择存储器地址，用地址线（P3[5：4]，P2[5：0]，P1[7；0]）。
6. 当前须写入的数据放入 P0[7：0]
7. 发脉冲到 ALE/PROG#，遵守最小脉冲宽度。
8. 等待 READY/BUSY# 由低转为高
9. 重复步骤 5-8 直到编程完毕。
10. 校验 FLASH 存储器的内容。

1.4.1.7 FLASH 存储器编程时序框图在外部主模式下



器件 ID=92H（SST89E516RD）
93H（SST89V516RD）
9BH（SST89E58RD）
9AH（SST89V58RD）

图 9-4-2： Read-ID

读芯片签名和识别寄存器在指定的地址（Read chip signature and identification register at the addressed location）

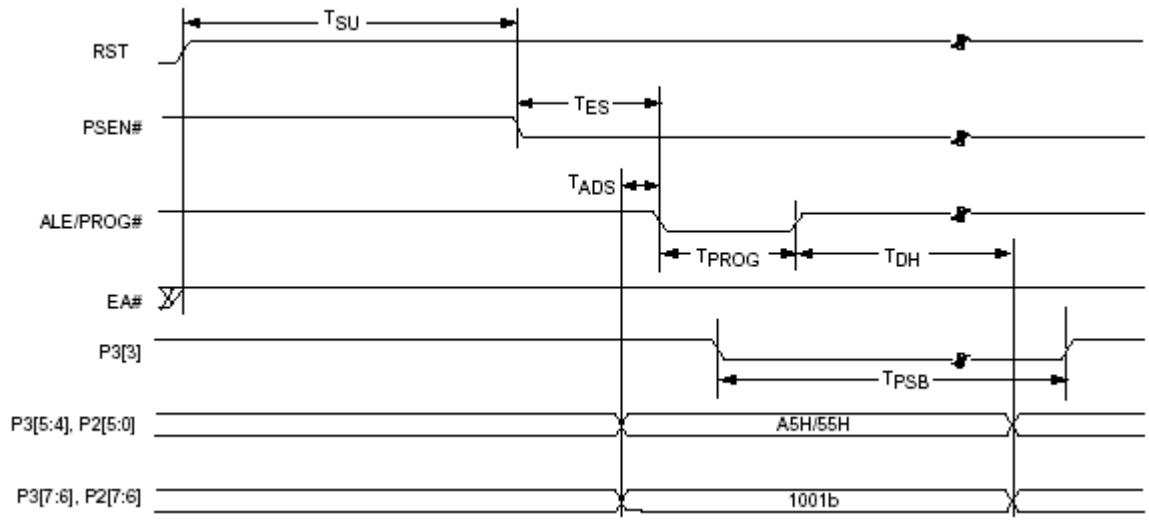


图 9-4-3： Select-Block1/ Select-Block 0

选择存储器两个块中的任何一个，在执行 Byte-Verify,Block-Erase,Sector-Erase 或

Byte-Program 命令前。这些命令仅适用于 SST89E516RD 和 SST89V516RD。

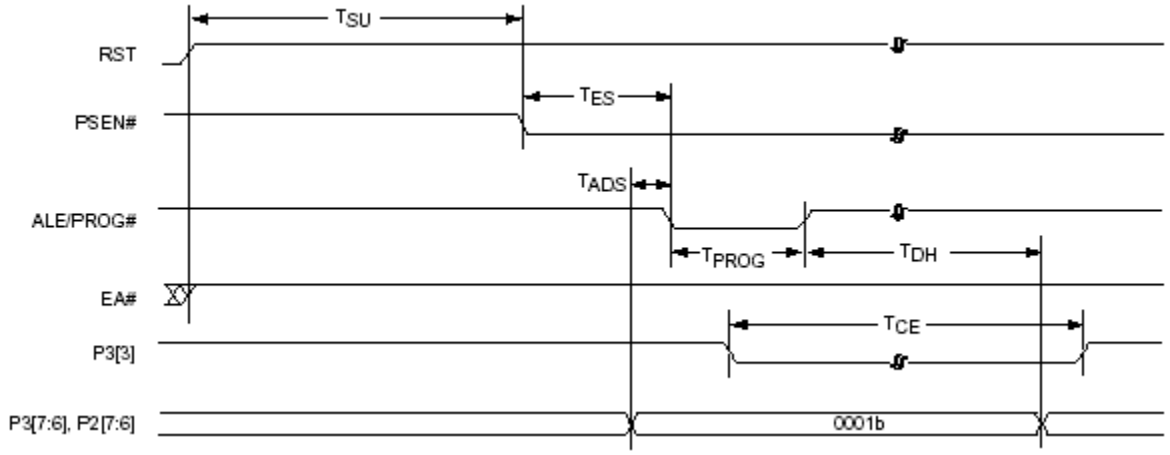


图 9-4-4 ： Chip–Erase（芯片擦除）

擦除两个 FLASH 存储块，加密锁忽略，加密位也被擦除

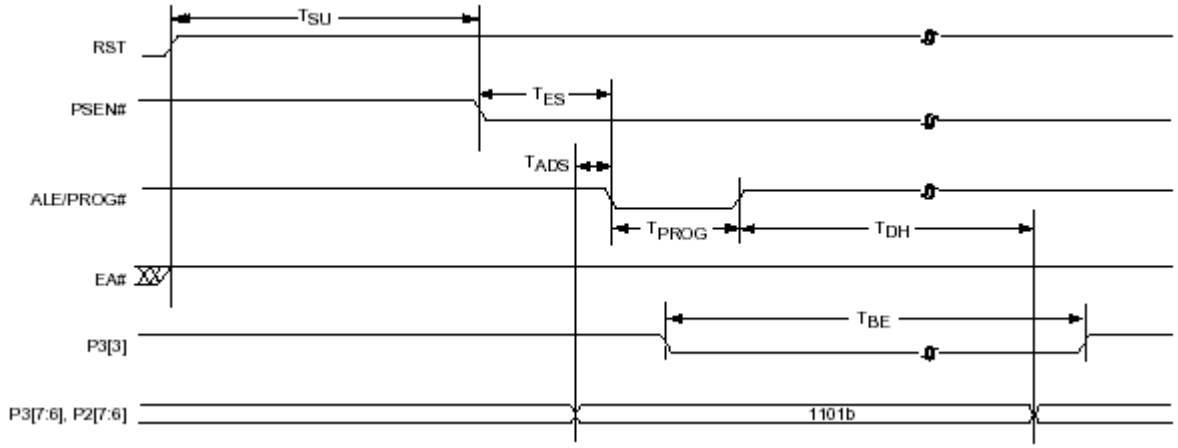


图 9-4-5：Block –Erase for SST89E/V516RD(块擦除)

擦除一个存储块，如果这一存储块的加密锁不被激活。

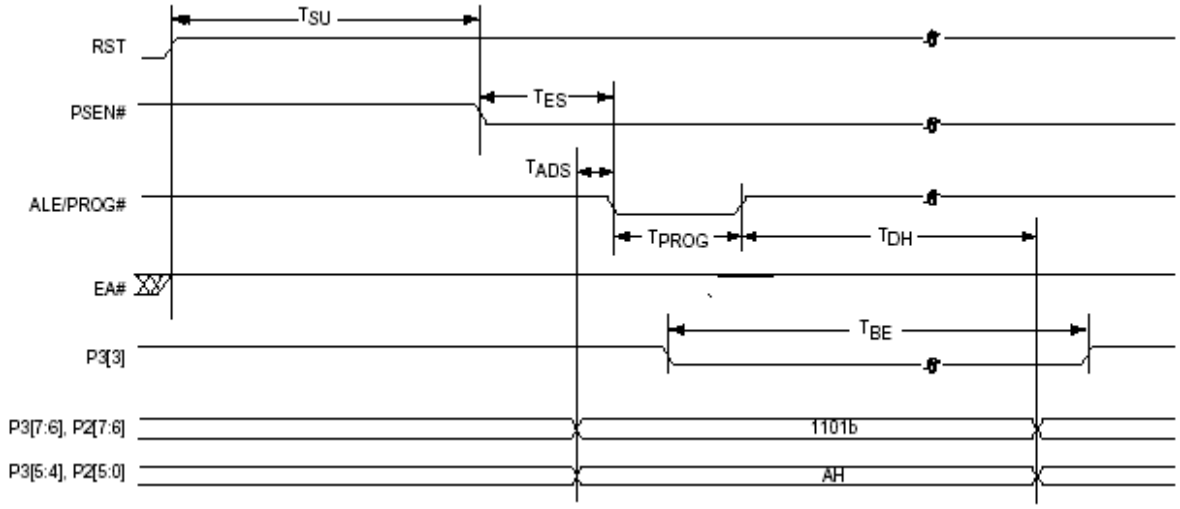


图 9-4-6： 块擦除 Block-Erase（SST89E/V52RD,54RD,58RD）

擦除一个存储块，如果这一存储块的加密锁不被激活。

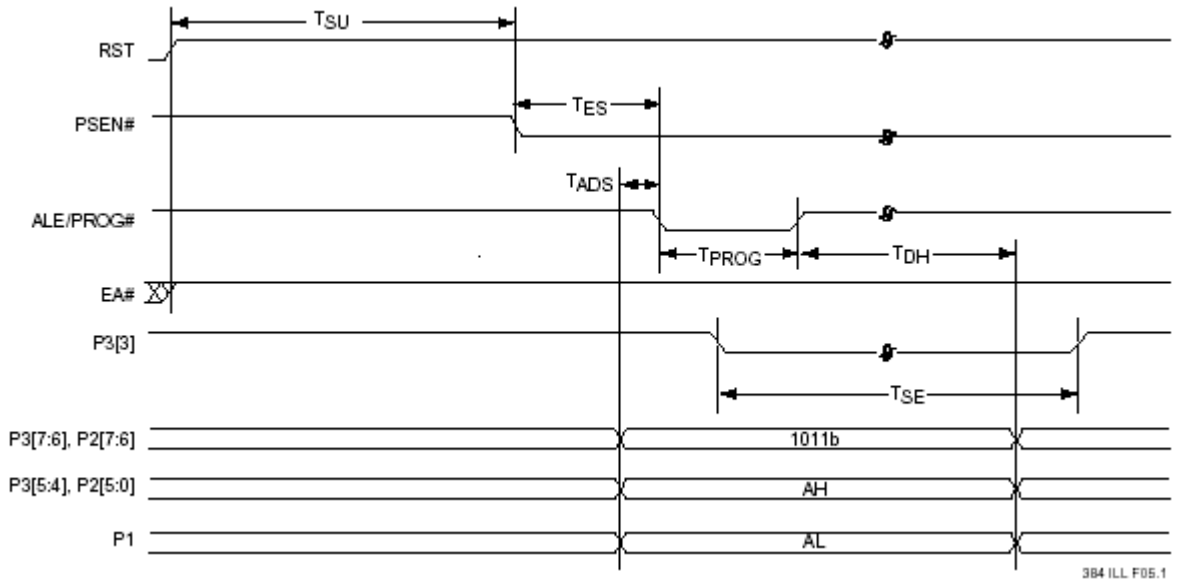


图 9-4-7： Sector-Erase(扇区擦除)

擦除指定扇区，如果这一存储块的加密锁不被激活。

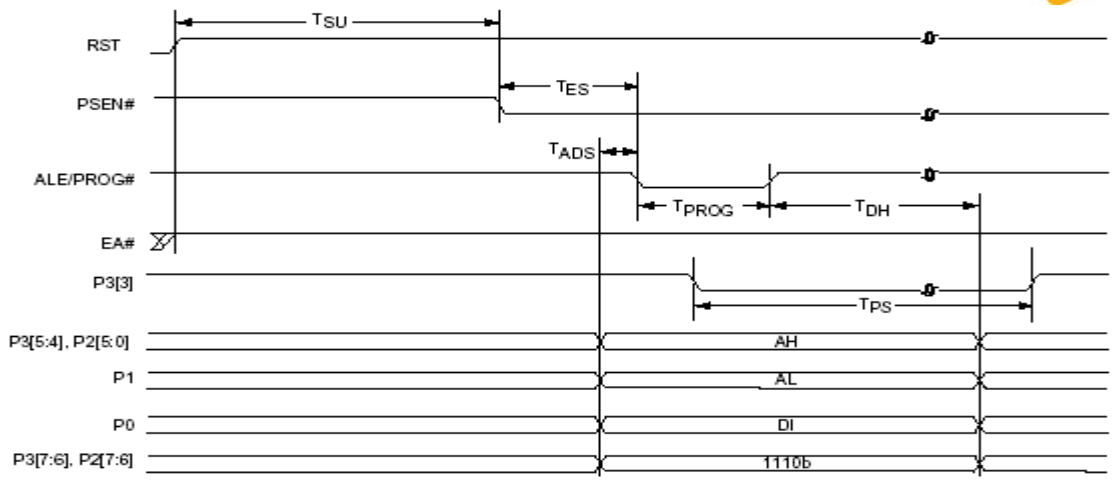


图 9-4-8：Byte-Program（字节编程）

编程指定代码字节 ，如果这个字节单元被成功擦除并且没有被编程。当那个 FLASH 存储块上的加密锁没有被激活时，字节编程才能被允许。

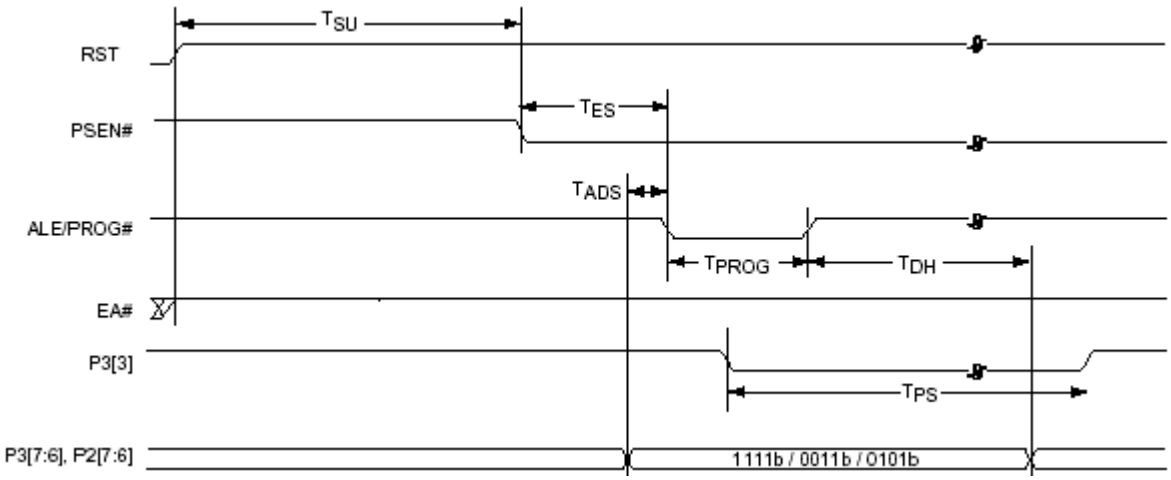


图 9-4-9 ：PROG-SB1/PROG-SB2/PROG/SB3

分别编程加密位 SB1、SB2 和 SB3。只有 CHIP-ERASE 能擦除可编程的加密位。

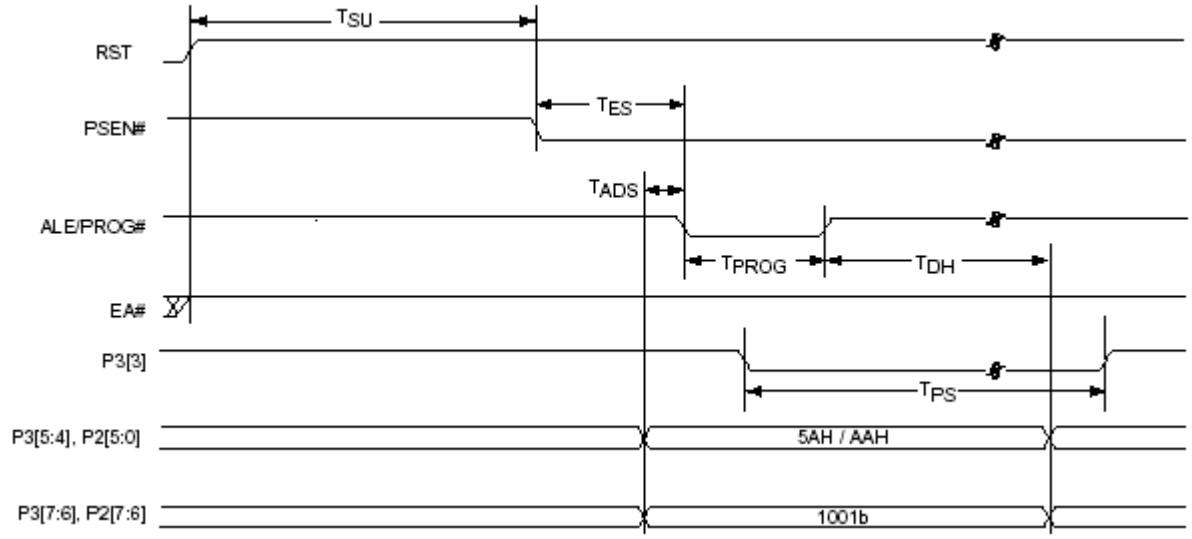


图 9-4-10：PROG-SC0/PROG-SC1

编程起动配置位 SC0/SC1。只有 Chip-Erase 才可擦除编程的 SC0/SC1 位。Prog-SC1 仅适用于 SST89E58RD 和 SST89V58RD。

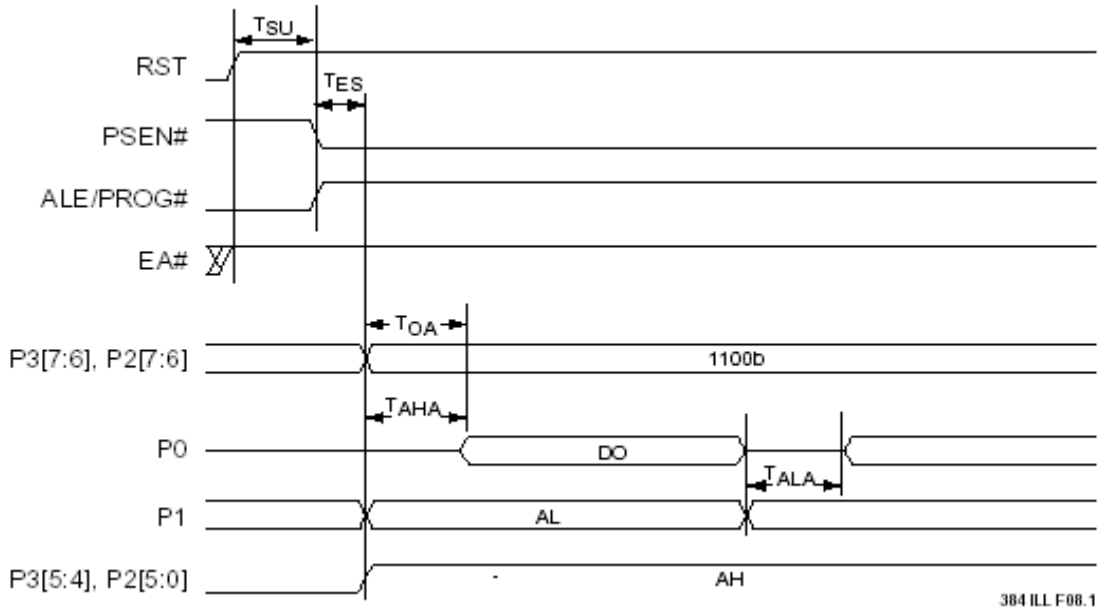


图 9-4-11： Byte-Verify（字节校验）

从 FLASH 存储器中的指定地址读代码字节，如果那一存储块的加密锁没有被激活。

1.4.2 IAP 模式（In-Application Programming Mode）

单片机提供 72K 或 40K 字节的在应用可编程的 FLASH 存储器。在 IAP 方式，微控制器的 CPU 进入 IAP 模式。FLASH 存储器的两个块允许 CPU 从一个块中执行用户代码，而同时另一个块被擦除或重新编程。当所有内部 FLASH 正在被重新编程时，CPU 也可从外部存储器取代码。在特殊功能寄存器中的邮箱寄存器（SFST，SFCM，SFAL，SFAH，SFDT 和 SFCF）控制和监测器件的擦除和编程过程。

表 9-4-6 概述了这些命令和相关邮箱寄存器的设置。

1.4.2.1 IAP 模式时钟源

在 IAP 模式下，CPU 和 FLASH 控制单元关闭了外部时钟，由内部振荡器为编程和擦除操作提供时钟参考。内部振荡器只有在需要时才启动，FLASH 操作完成时就立即关闭。

1.4.2.2 存储块选择(IAP 模式下)

寻址范围限制在 16 位，仅 64K 字节程序地址空间是可见的，在任何时候。如表 4-4 所示，块选择(EA#和 SFCF[1：0]的配置)允许 Block 1 覆盖 Block 0 的低 8K 字节，使 Block 1 可寻址。同样的概念被使用，允许 Block 0 和 Block 1 对 IAP 操作来说可进行。来自一个不可见块内的代码不能作为一个源去编程另一个地址。然而，一个不可见的块可以被另一个块的代码通过邮箱寄存器来编程。器件允许一个块内的 IAP 代码去编程另一个存储块，但不能在同一个块内编程任何单元。如果一个 IAP 操作物理上从 Block 1 开始，那么这个操作的目的地址隐式的定义为 Block 0。如果一个 IAP 操作物理上从 Block 0 开始，那么这个操作的目的地址隐式的定义为 Block 1。如果一个 IAP 操作从外部程序空间开始，那么它的目标地址将由块选择的状态和地址来决定。

1.4.2.3 IAP 允许位

IAP 允许位，SFCF[6]，允许 IAP 模式。只有该位置‘1’，所有的 FLASH 编程 IAP 命令才会被执行。

1.4.2.4 IAP 模式命令

所有下面的命令只能在 IAP 模式下被初始化。在任何情形下，写控制字到 SFCM 寄存器中将初始化所有的操作。如果加密锁对选择的块使能，则所有的命令都被禁止。

编程命令用来编程新数据到存储器阵列。存储阵列中将被编程的部分应该处于已擦除状态:FFH。如果存储器没有被擦除，首先调用相应的擦除命令擦除。警告：不要企图从一个

块中取代码又要写这个块，它将产生不可预料的编程行为和破坏程序数据。

Block-Erase 命令擦除两个存储块中任何一个块的所有字节。被擦除存储块的选择由 Block-Erase 命令的源来决定。如表 9-4-4 中所定义。

表 9-4-4 IAP 地址分配(SST89E516RD 和 SST89V516RD)

EA	SFCF[1：0]	IAP 指令的地址	目标地址	被编程的块
1	00	>=2000H（Block 0）	>=2000H （ Block 0）	无 [1]
1	00	>=2000H（Block 0）	<2000H（Block 1）	Block 1
1	00	<2000H（Block 1）	任何一个（Block 0）	Block 0
1	01，10，11	任何一个（Block 0）	>=2000H （ Block 0）	无
1	01，10，11	任何一个（Block 0）	<2000H（Block 1）	Block 1
0	00	从外部	>=2000H （ Block 0）	Block 0
0	00	从外部	<2000H（Block 1）	Block 1
0	01，10，11	从外部	任何一个（Block 0）	Block 0

注意：没有操作被执行,因为一个块的代码不能对自己编程.

Chip-Erase 命令擦除两个存储块中的所有字节。这个命令仅仅在 EA# =0 时被允许（执行外部存储器指令）。此外，当器件处于 4 级加锁时命令也不允许执行。在所有其它情况下，该命令将会忽略安全锁，并会擦除安全锁位和重新分配位。

Sector-Erase 命令擦除一个扇区内的所有字节。SST 的 MCU 内部的 FLASH 存储器扇区大小为 128 字节，选择哪一个扇区擦除由 SFAH 和 SFAL 的内容（即指定的地址）来决定。

Byte-Program 命令编程数据到一个单字节单元。地址由 SFAH 和 SFAL 的内容来决定。数据字节在 SFDT 特殊功能寄存器中。

Byte-Verify 命令让用户校验器件是否正确执行了一个擦除或编程命令。

如果命令成功执行，Byte-Verify 命令返回 SFDT 中的数据。在执行 Byte-Verify 命令前，用户须检查前一个 FLASH 操作是否执行完。Byte-Verify 命令的执行时间非常短，不必查询命令是否执行完，也没有中断产生。

Prog-SB3、Prog-SB2、Prog-SB1 命令用来编程安全位（参见表 8-1）。这些命令中的任何一个执行完后，加密选项将立即被更新。以前未被编程的加密位可通过这些命令来编程。Prog-SB3、Prog-SB2、Prog-SB1 命令只属于 Block 1。

Prog-SC0 命令用来编程 SC0 位。这个命令仅改变 SC0 位，对 BSEL 位没有影响，直到一个复位周期完成之后。以前未被编程的 SC0 位可通过这些命令来编程，Prog-SC0 只属于 Block 1。

Prog-SB1 命令用来编程 SC1 位，这个命令仅改变 SC1 位，对 BSEL 没有影响，直到一个复位周期完成之后。以前未被编程的 SC1 位可通过这些命令来编程，Prog-SB1 命令只属于 Block 1。

没有与外部主命令 Select-Block0 和 Select-Block1 命令相对应的 IAP 命令。

1.4.2.5 查询

用查询方法来检测 FLASH 操作是否完成的命令应该查询 FLASH-BUSY 位（SFST[2]）。当 FLASH _BUSY 为“0”时， 器件为下一操作作好了准备。

MOVC 指令也可用来检测 FLASH 存储器的编程和擦除操作。如果 MOVC 指向的那一存储块处于忙状态，那么执行 MOVC 指令就会失败。

1.4.2.6 中断结束（当 FLASH 操作完成产生中断）

如果中断结束（SFCM[7]置位）被选择，那么一个中断（INT1）将产生，指出 FLASH 操作已完成。 在这种情形下，INT1 成为一个内部中断源。INT1# 引脚（P3_3）现在可用作为一个通用功能口，并且在 IAP 模式下，它不能用作外部中断 1 的中断源。

为了用一个中断来表明一FLASH 操作结束，IE 寄存器的EX1 和EA 位必须置位，TCON 寄存器的 IT1 位也必须置位，来设置为一个边沿触发检测中断。

表 9-4-5 IAP 模式命令[1] (SST89E/V516RD)

操 作	SFCM[6：0][2]	SFDT[7：0]	SFAH[7：0]	SFAL[7：0]
Block-Erase[3]	0DH	55H	X[4]	X
Sector-Erase[3]	0BH	X	AH[5]	AL[6]
Byte-Program[3]	0EH	DI[7]	AH	AL
Byte-Verify(Read)[3]	0CH	DO[8]	AH	AL
Prog-SB1[9]	0FH	AAH	X	X
Prog-SB2[9]	03H	AAH	X	X
Prog-SB3[9]	05H	AAH	X	X
Prog-SC0[9]	09H	AAH	5AH	X

1. SFCF[6] =1 ： 允许 IAP 命令；SFCF[6]=0： IAP 命令禁止
2. 中断/查询允许，FLASH 操作完成时
   SFCM[7]=1 中断允许
   SFCM[7]=0 选择允许
3. 参考表 4-4 中关于地址的分配
4. X 可能是 V_{IL} (低)或 V_{IH}（高），但不是其它值
5. AH=地址高 8 位
6. AL=地址低 8 位
7. DI=数据输入
8. DO=数据输出
   所有其它的值是 16 进制数。
9. 命令必须在 Block 1

表 9-4-6 IAP 模式命令[1] (SST89E/V52RD,54RD,58RD)

操 作	SFCM[6：0][2]	SFDT[7：0]	SFAH[7：0]	SFAL[7：0]
Block-Erase[3]	0DH	55H	AH[4]	X[5]
Sector-Erase[3]	0BH	X	AH[6]	AL[7]
Byte-Program[3]	0EH	DI[8]	AH	AL
Byte-Verify(读)[3]	0CH	DO[9]	AH	AL
Prog-SB1[10]	0FH	AAH	X	X
Prog-SB2[10]	03H	AAH	X	X
Prog-SB3[10]	05H	AAH	X	X
Prog-SC0[10]	09H	AAH	5AH	X
Prog-SC1[10]	09H	AAH	AAH	X

1．SFCF[6] =1 IAP 命令允许；SFCF[6]=0 IAP 命令禁止

2．表明 FLASH 操作完成的中断/选择允许

SFCF[7]=1 中断允许

SFCF[7]=0 选择允许

3．可参阅表 4-4 中关于地址的分配

4．SFAH[7]=0 选择 Block 0； SFAH[7]=1 选择 Block 1

5．X 可是 V_{IL} 或 V_{IH} ，但不能是其它值

6．AH 地址高 8 位

7．AL 地址低 8 位

8．DI 数据输入

9．DO 数据输出

10．命令必须在 Block 1 中

表 9-4-7 FLASH 存储器的编程/校验参数

Parameter[1,2]	Symbol	Min	Max	Units
Reset Setup Time	T_{SU}	3		μs
Read-ID Command Width	T_{RD}	1		μs
PSEN# Setup Time	T_{ES}	1.125		μs
Address, Command, Data Setup Time	T_{ADS}	0		ns
Chip-Erase Time	T_{CE}		125	ms
Block-Erase Time	T_{BE}		100	ms
Sector-Erase Time	T_{SE}		30	ms
Program Setup Time	T_{PROG}	1.2		μs
Address, Command, Data Hold	T_{DH}	0		ns
Byte-Program Time[3]	T_{PB}		50	μs
Select-Block Program Time	T_{PSB}		500	ns
Security bit Program Time	T_{PS}		80	μs
Verify Command Delay Time	T_{OA}		50	ns
Verify High Order Address Delay Time	T_{AHA}		50	ns
Verify Low Order Address Delay Time	T_{ALA}		50	ns

1. 编程/擦除时间反过来按比例衡量编程时钟频率
2. 所有时序测量是从输入的中间到输出的中间
3. 每一个字节在编程前必须擦除

1.5 定时器/计数器

器件有三个 16 位寄存器，可以用来作为定时器或计数器。这三个定时器/计数器分别是 Timer 0（T0）、Timer 1（T1）、Timer 2（T2）。每个都被设计成特殊功能寄存器（SFR）中的一对 8 位寄存器。这些寄存器分别为 TL0、TH0，TL1、TH1，TL2 和 TH2。

1.6 串行输入/输出（SERIAL I/O）

1.6.1 增强型全双工异步串行通信（UART）

器件的串行输入输出是全双工的，允许同时接收和发送数据通过各自的接收和发送寄存器，而同时软件可以执行其它的任务。发送和接收寄存器都位于特殊功能寄存器中的串行数据缓冲器 SBUF 中。写数至 SBUF 等于向发送寄存器装载数据，从 SBUF 读数据相当于获得了接收寄存器中的数据。

UART 有四种工作方式，它们能通过特殊功能寄存器 SCON（Serial Port Control）的 SM0 和 SM1 来选择。在四种方式中，发送通过一个将 SBUF 寄存器作为目标寄存器的指令开始。在方式 0，当 SCON 的接收中断标志位 RI 清零和 SCON 的允许接收位 REN 置 1 时，接收开始。在其它方式下，如果 SCON 的 REN 置 1，那么接收从接收到开始位开始。

1.6.1.1 帧错误检测

帧错误检测允许串行口自动检查有效停止位，在方式 1、方式 2 和方式 3 下。如果停止位丢失，帧错误标志位（FE）将置‘1’，软件然后能检查这个位，在一个检查到有通信错误的接收之后。FE 必须通过软件清零。

FE 位在 SCON 中，与 SM0 共享一个位地址。PCON 的 SMOD0 位决定了两个位中哪一个被访问。当 SMOD0=0 时，SCON[7]=SM0，当 SMOD0=1 时，SCON[7]=FE。

1.6.1.2 自动地址识别（Automatic Address Recognition）

自动地址识别（AAR）减少了 CPU 在多机通信环境中服务串行口的时间。当使用 AAR 时，串行口硬件只有在接收到它自己的地址时才产生一个中断，这样就减少了需要比较地址的软件消耗。

AAR 只有在方式 2 或方式 3 下使用串行口时才有效。置位 SCON 的 SM2 位可允许 AAR。当在等待一地址（第九位为 1）时从机必须将 SM2 置‘1’。接收中断标志（RI）只有在接收到的字节与给定地址或广播地址相匹配时才置 1。然后从机清 SM2 允许从主机接收数据（第九位为 0）。

主机能通过发送给定地址与多个从机进行选择性通信。还可通过发送广播地址寻址所有从机。

SADDR 和 SADEN 是为这些从机定义地址的特殊功能寄存器。SADDR 定义从机独立的地址,SADEN 是一个屏蔽字节,当与 SADDR 结合，用来定义不需要注意的位，形成给定的地址。下面是一个例子:

UART 从机 1		
SADDR	=	1111 0001
SADEN	=	1111 1010
GIVEN	=	1111 0x0x
UART 从机 2		
SADDR	=	1111 0011
SADEN	=	1111 1001
GIVEN	=	1111 0xx1

在这个例子中， 可以用位 0 和位 1 来区别从机 1 和从机 2。如果位 1 为 1，从机 1 不会响应,而从机 2 可以。同样地，如果位 0 为 0，从机 2 就不会响应,而从机 1 可以。两个从机都会响应 11110x01b 的地址，这就是一个广播地址。可通过 SADDR 和 SADEN 来形成广播地址，‘0’为不需要注意的位。

1.6.2 SPI（Serial Peripheral Interface）

器件的 SPI 可与同类器件的 SPI 或其它兼容 SPI 器件进行高速全双工同步数据传送。

图 9-6-1 展示了主和从 SPI 器件之间的数据通信。SCK 引脚是主/从模式下的时钟输出/输入。在向主机的 SPI 数据寄存器写数后，SPI 时钟发生器启动，这样被写入的数据就从主机的 MOSI 输出，进入从机的 MOSI。在完成一字节的数据传送后，SPI 时钟发生器停止、SPIF 标志位置位。SPI 中断请求将产生，如果 SPI 中断允许位（SPIE）和串行中断允许位（ES）都被置位的话。外部主机驱动从机选择输入脚 SS#/P1[4]为低来选择 SPI 模块作为从机。如果 SS#/P1[4]没有置为低，从机 SPI 单元将不被激活，MOSI/P1[5]仍可用作普通输入引脚。

CPHA 和 CPOL 控制 SPI 时钟的相位和极性。图 9-6-2 和 9-6-3 列出了这两位可能的四种组合。

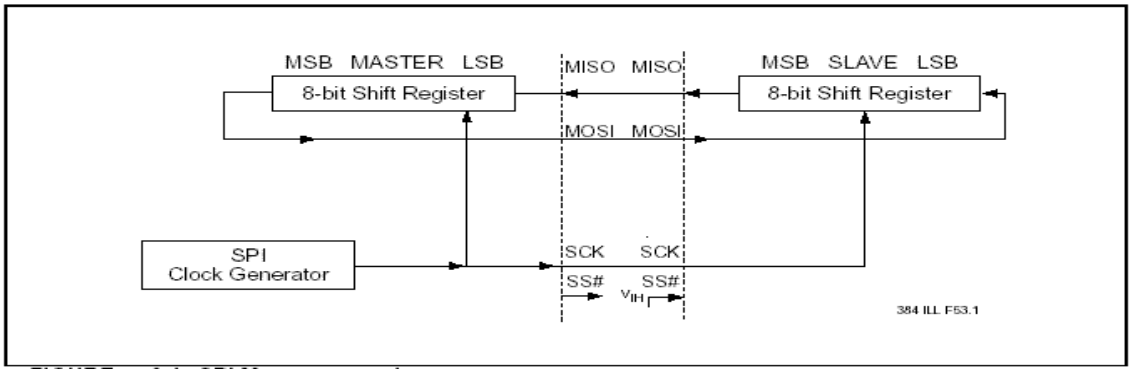


图 9-6-1：SPI 主-从连接

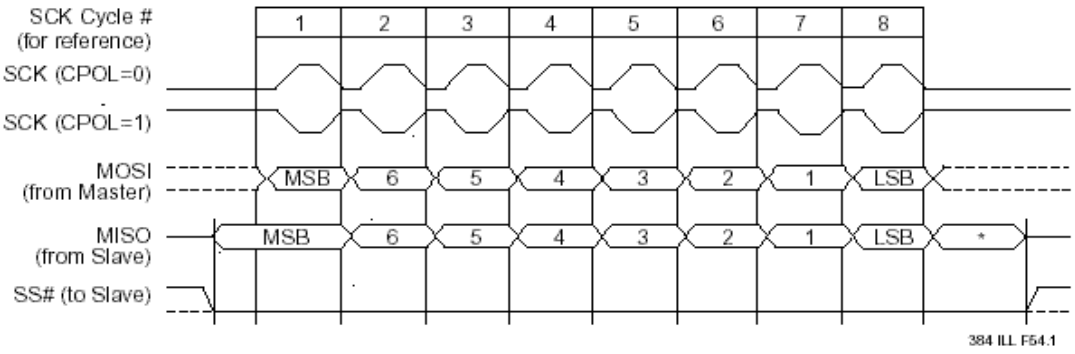


图 9-6-2 SPI 通信格式（CPHA=0）

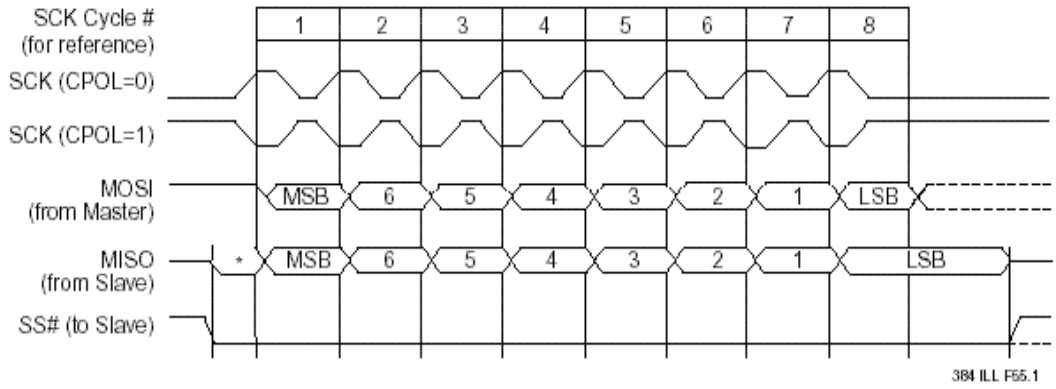


图 9-6-3 SPI 通信格式（CPHA=1）

1.7 看门狗定时器（WATCHDOG TIMER）

器件提供一个可编程的看门狗定时器（WDT），以防止进入软件死锁并能自动恢复。为了防止系统进入软件死锁，用户软件必须在用户定义的周期内刷新 WDT 。如果软件没有将 WDT 周期性刷新，内部硬件复位将产生，如果允许（WDRE=1）的话。如果程序没有正确执行，看门狗定时器溢出。器件内的 WDT 用系统的时钟（XTAL1）来作它的时钟基准。这样，严格来说，它是看门狗计数器而不是看门狗定时器。每 344064 个时钟，WDT 寄存器加 1。定时基数寄存器（WDTD）的高 8 位用作 WDT 的重装寄存器。如果 WDT 溢出，WDTS 标志位置位，WDT 复位不能改变这个标志位。用户软件能向它写“1”来清除 WDTS。

图 9-7-1 是 WDT 的框图。两个 SFRs（WDTC 和 WDTD）控制看门狗定时器的工作。在 Idle 模式，WDT 操作被临时挂起，在接收到一个中断从 Idle 退出，操作将会被恢复。WDT 的溢出时间是如下计算的： 周期=（255-WDT）*344064/f_{OSC}。这里，WDT 是装入 WDT 寄存器的值，f_{OSC} 是振荡器频率。

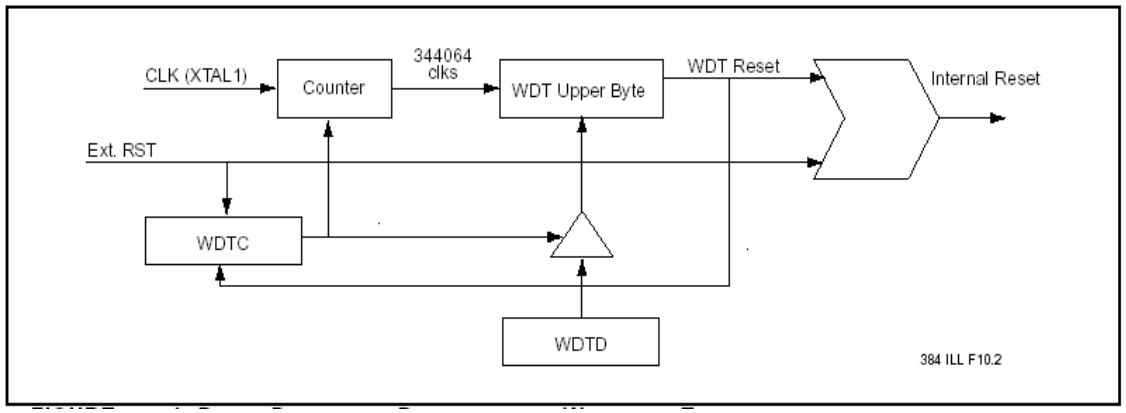


图 9-7-1：可编程看门狗定时器框图

1.8 可编程计数器阵列（PCA）

器件配有一个完整的程序计数阵列（PCA）。PCA 配备一个专门的定时/计数器，该定时/计数器用作 5 个比较/俘获模块的通用时间基础。每个模块能编程为 4 个模式中的一个。除此

之外，第五个模块能编程为一个看门狗定时器。

1.8.1 PCA 定时/计数器

PCA 的定时/计数器是一个自由运行的 16 位寄存器，由寄存器 CH 和 CL 组成（计数值的高 8 位和低 8 位）。这些寄存器能在任何时刻进行读和写。CMOD 寄存器的计数脉冲选择位（CPS1 和 CPS0）设置定时/计数器的四种模式。参考表 9-8-1。CMOD 寄存器还包含计数 Idle（CIDL）位。当 CIDL=1，并且 MCU 进入 Idle 模式时，PCA 定时/计数器关闭。

表 9-8-1： 计数脉冲选择位

CPS1	CPS0	PCA 计数脉冲选择
0	0	内部时钟，FOSC/12
0	1	内部时钟，FOSC/4
1	0	定时器 0 溢出
1	1	通过 P1.2 输入的外部时钟

寄存器 CCON 内的计数器运行位（CR）控制定时/计数器的开和关。当 CR=1 时，定时/计数器正在工作，当 CR=0 时，定时/计数器禁止。当 PCA 定时/计数器溢出 CCON 的 CF 位将被置位，如果 CMOD 的 ECF 置位，将会产生一个中断。

1.8.2 PCA 比较/俘获模块

每个比较/俘获模块有一个称为 CCAPMn(n=0,1,2,3,4)的模式寄存器，每一个选择它将实现的功能。CCAPMn 的七个可能的模式和它们的相关值如表 9-8-2 所示。

表 9-8-2：CCAPMn 的可能模式和相关值

	CCAPMn 的值	
模块功能	没有中断允许	中断允许
只俘获上升沿	20H	21H
只俘获下降沿	10H	11H
俘获上升沿和下降沿	30H	31H
16 位软定时器	48H	49H
高速输出	4CH	4DH
脉冲宽度调节器	42H	43H
看门狗定时器 [1]	48H 或 4CH	-

1 只用于模块 4

每个模块有一个 8 位俘获/比较寄存器（CCAPnH 和 CCAPnL）和一个相关的外部输入/输出引脚。外部输入/输出引脚是 P1.3(对模块 0)，P1.4(对模块 1)，P1.5(对模块 2)，P1.6(对模

块 3)，P1.7(对模块 4)。每个模块有一个位于 CCON 寄存器的相关事件标志 CCFn。这些标志必须通过软件清零。写入到 CCAPnL 将禁止相应模块的比较特性，写入到 CCAPnH 将重新允许它。而且，当用到比较特性（16 位软定时器，高速输出，脉冲宽度调节器和看门狗定时器模式），程序总是先写入 CCAPnL 然后再写 CCAPnH.

1.8.2.1 俘获模式

俘获模式用来获取 PCA 定时/寄存器的值并保存到模块的俘获寄存器（CCAPnH 和 CCAPnL）。俘获将发生在相关外部输入引脚的输入信号的上升沿、下降沿或上升沿/下降沿，这依赖于哪个模式被选择。事件标志（CCFn）和 ECCFn 置位将产生一个中断。

1.8.2.2 16 位软定时器模式

在 16 位软定时模式，PCA 定时/计数器的值用来与预先装入模块比较寄存器（CCAPnH 和 CCAPnL）的 16 位值相比较。当匹配时，事件标志（CCFn）置位，如果 ECCFn 也被置位的话，将产生一个中断。

1.8.2.3 高速输出模式

在高速输出模式，PCA 定时/计数器的值用来与预先装入模块比较寄存器（CCAPnH 和 CCAPnL）的 16 位值相比较。当匹配时，模块相应的输出引脚被触发。事件标志（CCFn）置位，如果 ECCFn 也被置位的话，将产生一个中断。输出频率由 PCA 定时/计数器决定并且对 5 个模块都是一样的，但是占空系数的改变依赖于预先装入到比较寄存器内的值。

1.8.2.4 脉冲宽度调节器模式

脉冲宽度调节器通过比较 PCA 定时器的低 8 位（CL）和比较寄存器的低 8 位(CCAPnL)来产生一位 PWMs 。当 CL〈 CCAPnL，相应的输出引脚是低。当 CL 〉 CCAPnL，相应的输出引脚是高。PWM 的频率由 PCA 定时/计数器决定并且对 5 个模块都是一样的，DUTY 周期的改变将依赖于 CCAPnL 内的值。CCAPnL 能通过装入一个新值到 CCAPnH 来动态改变。当 CL 从 FFH 滚动到 00H 时，新值就被移进了 CCAPnL。

1.8.2.5 看门狗定时器模式

只有模块 5 能编程为一个看门狗定时器（如果看门狗定时器没有用上的话它也能编程为其它模式）。看门狗定时器比较 PCA 定时/计数器的值（CH 和 CL）和模块 4 的比较寄存器（CCAP4H 和 CCAP4L）。

当匹配时，如果 CMOD 寄存器的 WDTE 位置位，将产生一个内部复位。这个内部复位不会将 RST 引脚驱动为高。为了保持在复位状态，用户必须事先改变比较值，这样它就永远不会与 PCA 定时器匹配。

1.9 加密锁

加密锁防止软件盗版，也防止非法用户读取 FLASH 内容。它也防止由于对 FLASH 存储器的意外擦除和编程操作而产生代码异常。单片机的加密锁系统有两个不同类型的加密锁 ：硬件锁和软件锁。

1.9.1 硬件锁

当硬件锁被激活时，从未加密或由软件加密程序地址空间执行的 MOVC 或 IAP 指令不允许从硬件加密存储块中读取代码字节（参考表 9-8-2）。硬件锁可锁住两个 FLASH 存储块或只锁住 FLASH 存储块（Block 1）的 8K 字节。除了 Chip-Erase，所有的外部主命令和 IAP 命令对硬件锁存储块来说都被忽略。

1.9.2 软件锁

在安全环境下，软件锁允许 FLASH 内容被改变。在预先设定的可靠环境下，锁的选项允许用户通过 IAP 模式来更新被软件加密的存储块内的程序代码。例如，如果 Block 1（8K）被加密（软件加密或硬件加密），Block 0（64K ）是软件加密的，Block 1 中的代码编程 Block 0。下面是通过命令邮箱寄存器（SFCM）发出，从一个加密（软件加密或硬件加密）存储块执行，能对一软件加密存储块操作的 IAP 命令：Block-Erase、Sector-Erase、Byte-Program 和 Byte-Verify。

在外部主模式，软件锁等同于硬件锁。

1.9.3 加密锁状态

SFST[7：5]这三位表示器件的加密锁状态。如图 9-8-1 和表 9-8-1 所示，三个加密锁位控制存储器初级和次级存储块的加密状态。加密锁状态有四个不同的等级。在第一级，没有加密锁位被编程，两个块都没上锁。在第二级，虽然两个块都被锁住不能被编程，但能通过 Byte-Verify 进行读操作。在第三级，有下面三个可选择的状态：Block 1 硬件锁/Block 0 软件锁，软件锁住两个块，硬件锁住两个块。除了读操作不可用外，锁定两个块与第二级别是一样的。第四级是加密级别最高的一级，它不允许对内部存储器进行读和编程操作，也不允许从外部存储器启动。请注意没有用到的加密锁位的组合，器件将默认为第四级。关于如何编程加密锁位的详细说明可参考外部主模式和 IAP 部分章节。

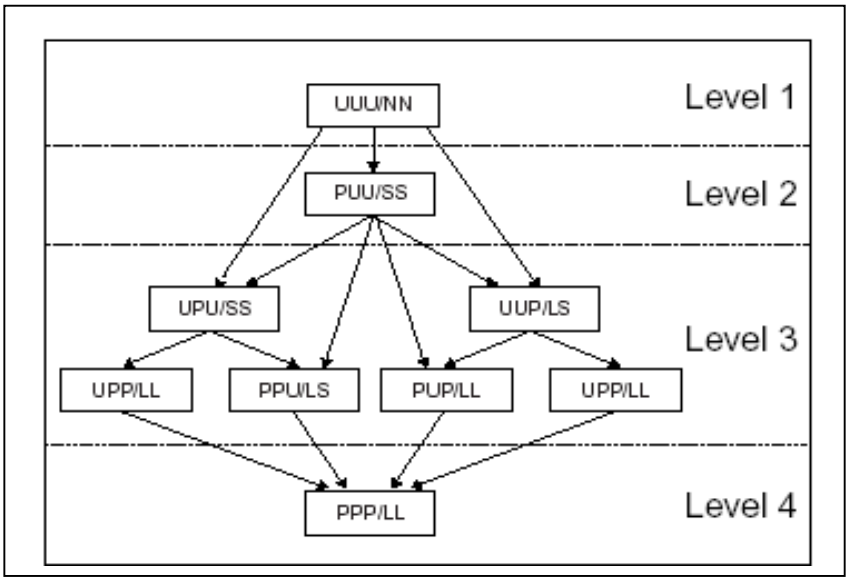


图 9-8-1：安全加密等级

注意： P= Programmed(Cell logic state=0),U=Unprogrammed(Cell logic state=1), N=Not Locked,L=Hard, Locked,S=SoftLocked.

表 9-8-1 ：加密锁可选项

级别	加密锁位				加密状态		加密类型
	SFST[7：5]	SB1	SB2[1]	SB3[1]	Block 1	Block 0	
1	000	U	U	U	不加密	不加密	没有加密特性被允许。
2	100	P	U	U	软件锁	软件锁	从外部程序存储器开始执行的 MOVC 指令不能从内部存储器取代码，EA#就是一个例子，锁定 Reset,对 flash 存储器的进一步编程不被允许。
3	011 101	U P	P U	P P	硬件锁	硬件锁	级别 2 加上禁止校验，两个块都锁住
	010	U	P	U	软件锁	软件锁	级别 2 加上禁止校验，Block 1 中的代码能编程 Block 0，反之亦然。
	110 001	P U	P U	U P	硬件锁	软件锁	级别 2 加上禁止校验，Block 1 中的代码能编程 Block 0。
4	111	P	P	P	硬件锁	硬件锁	同级别 3 的硬件锁/硬件锁一样，但是 MCU 的指令从内部存储器开始执行，不管 EA#如何。

1．P=Programmed （Cell logic state=0 ）， U=Unprogrammed （Cell logic state=1）

2. SFST[7:5]=Security Lock Decoding Bits（SECD）

表 9-9-2：加密锁访问表

级别	SFST[7：5]	源地址	目标地址 [1]	外部主机 Byte-Verify 允许 [2]	IAP Byte-Verify 允许	MOVC 指令允许（on564）	MOVC 指令允许（on554）
4	111b (硬件锁住两个块)	Block0/1	Block0/1	N	N	Y	Y
			外部	N/A	N	N	N
		外部	Block0/1	N	N	N	N
			外部	N/A	N	N	N
3	011b/101b (硬件锁住两个块)	Block 0/1	Block0/1	N	N	Y	Y
			外部	N	N	N	Y
		外部	Block0/1	N	N	N	N
			外部	N/A	N	Y	Y
	001b/110b (Block 0=软件锁，Block 1=硬件锁）	Block 0	Block 0	N	N	Y	Y
			Block 1	N	N	N	N
			外部	N/A	N	N	Y
		Block 1	Block 0	N	Y	Y	Y
			Block 1	N	N	Y	Y
			外部	N/A	N	N	Y
		外部	Block0/1	N	N	N	N
			外部	N/A	N	Y	Y
	010b (软件锁住两个块)	Block 0	Block 0	N	N	Y	Y
			Block 1	N	Y	Y	Y
			外部	N/A	N	N	Y
		Block 1	Block 0	N	Y	Y	Y
			Block 1	N	N	Y	Y
			外部	N/A	N	N	Y
		外部	Block0/1	N	N	N	N
			外部	N/A	N	Y	Y
2	100b (软件锁住两个块)	Block 0	Block 0	Y	N	Y	Y
			Block 1	Y	Y	Y	Y
			外部	N/A	N	N	Y
		Block 1	Block 0	Y	Y	Y	Y
			Block 1	Y	N	Y	Y
			外部	N/A	N	N	Y
		外部	Block0/1	Y	N	N	N
			外部	N/A	N	Y	Y
	000b (不上锁)	Block 0	Block 0	Y	N	Y	Y
			Block 1	Y	Y	Y	Y
			外部	N/A	N	N	Y

		Block 1	Block 0	Y	Y	Y	Y
			Block 1	Y	N	Y	Y
			外部	N/A	N	N	Y
		外部	Block0/1	Y	Y	N	Y
			外部	N/A	N	Y	Y

1．MOVC 指令地址。

2．外部主机 Byte-Verify 存取不依靠源地址。

1.10 复位（RESET）

系统复位初始化 MCU，程序从程序存储器的 0000H 开始执行。器件的复位输入是 RST 引脚 。为了复位器件，在振荡器稳定后，必须在 RST 引脚致少加两个机器周期（24 个时钟）的逻辑高电平。复位期间， ALE 和 PSEN#被弱上拉。在复位时，ALE 和 PSEN#输出高电平来表示成功的复位，这个电平一定不能被外界因素影响。在器件运行过程中，系统复位不会影响片内 1K 字节的 RAM。而在上电时，片内 RAM 的内容是不确定的。复位后，所有的特殊功能寄存器（SFR）都返回到表 9-3-5 到 9-3-9 所示的复位值。

1.10.1 上电复位（Power-on Reset）

在最初上电时，所有端口处于随机状态，直到振荡器启动，内部复位弱上拉所有的端口。没有一有效的复位而对器件加电可能导致 MCU 从一个随机的地址开始执行指令。这些不确定的状态可能在不经意间就破坏了 FLASH 中的代码。

电源供电后， RST 引脚必须保持足够长时间的高电平直到振荡器稳定（对低频率晶体来说通常是几个毫秒），除了有效的加电复位是两个机器周期外。一个扩展 RST 信号的方法是在 V_{DD} 端接一 10μF 的电容，在 V_{SS} 端接一 8.2K 的电阻来产生一 RC 电路，如图 9-10-1 所示。要注意的是，如果用 RC 电路，必须保证 V_{DD} 的上升时间不能超过 1mS，振荡器起振时间不超过 10mS 。

对有较长起振时间的低频率振荡器来说，必须扩展复位信号来解决这个问题。这个方法保证了 V_{DD} 和 RST 间的必要关系来避免程序从不确定的地址开始执行，它可能破坏 FLASH 内的代码。要获得关于系统设计技术的更多信息，请参阅《SST FlashFlex51 系列单片机应用说明书》。

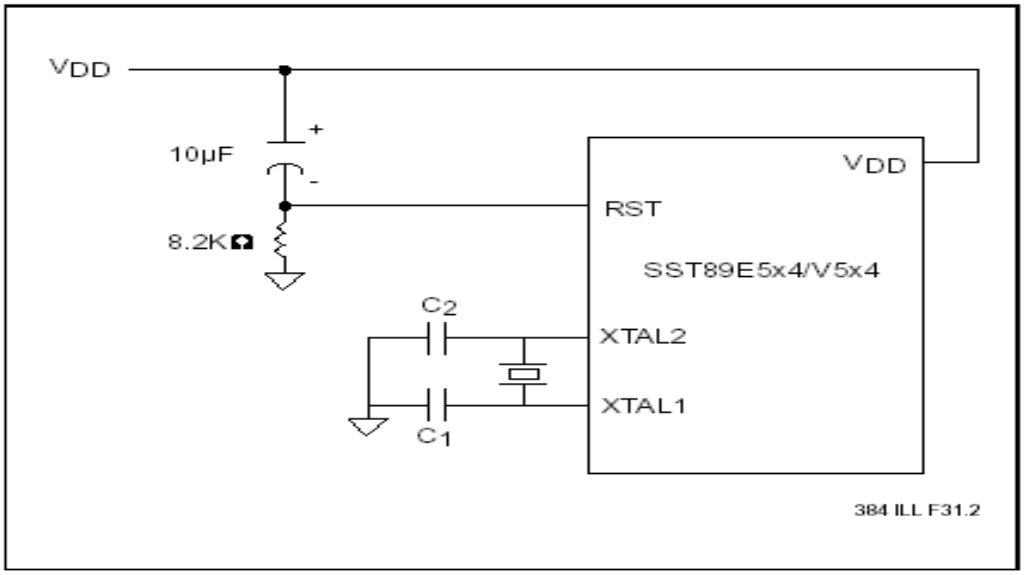


图 9-10-1： 上电复位电路

1.10.2 软件复位（Software Reset）

软件复位是通过将 SFCF[1]（SWR）由‘0’改为‘1’来执行的。软件复位将程序计数器（PC）复位到 0000H。所有的 SFR 寄存器都置为它们的复位值，除了 SFCF[1]（SWR）、WDTC[2]（WDTS）， RAM 的数据将不被改变。

1.10.3 掉电检测复位(Brown-out Detection Reset)

器件包含一个 Brown-out 检测电路来防止严重的 V_{DD} 抖动。Brown-out 电压参数请参考表 9-10-3 和 9-10-4。置位 IEA 寄存器（Address E8H,bit3）的 EBO 位允许 Brown_out 中断。如果 EBO 被置位，Brown_out 条件发生，那么一个 Brown_out 中断将会产生,中断程序从 004BH 开始执行。Brown_out 中断响应后，需通过软件清零 EBO。在 Brown_out 条件激活的情况下清零 EBO 会复位器件。如果 Brown_out 中断禁止，Brown_out 条件将复位程序从地址 0000H 处重新开始执行。

1.10.4 中断优先级和查询顺序

器件有 8 个中断源,四个中断优先级别。 表 9-10-1 概括了器件所支持中断的查询顺序。需要指出的是 SPI 串行接口和 UART 共享同样的中断入口地址。

表 9-10-1： 中断查询顺序

类型	中断标志	向量地址	中断允许	中断优先级	响应顺序	唤醒掉电模式
Ext.Int0	IE0	0003H	EX0	PX0/H	1(最高)	能
Brown-out	BOF	004BH	EBO	PBO/H	2	不能
T0	TF0	000BH	ET0	PT0/H	3	不能
Ext.Int1	IE1	0013H	EX1	PX1/H	4	能
T1	TF1	001BH	ET1	PT1/H	5	不能
UART/SPI	TI/RI/SPIF	0023H	ES	PS/H	6	不能
T2	TF2,EXF2	002BH	ET2	PT2/H	7	不能

1.10.5 省电模式(Power-Saving Modes)

器件提供三种省电模式供应用，当电源消耗需要考虑时。这三种省电模式是：空闲，掉电和 Standby 模式。

1.10.5.1 空闲模式（Idle）

当 PCON 寄存器的 IDL 置位时进入 Idle 模式。Idle 模式下，程序计数器（PC）停止。在这种模式下系统时钟继续运行，所有的中断和外围设备都是激活的。片内 RAM 和特殊功能寄存器保持原有的数据。通过系统中断或硬件复位可退出 Idle 模式。通过系统中断退出 Idle 模式，在中断的开始就清零 IDL 位，退出 Idle 模式。退出中断服务程序后，被中断程序紧接着从引起 Idle 模式命令的后一指令重新开始执行。硬件复位同上电复位一样都能启动器件。

1.10.5.2 掉电模式（Power Down Mode）

将 PCON 寄存器的 PD 位置位,器件进入掉电模式。在掉电模式下，时钟停止，仅对电平敏感的外部中断有效。保持片内 RAM 的内容和所有特殊寄存器的值，V_{DD} 端的最小电平应保证在 2.0 伏。器件通过允许对电平敏感的外部中断和硬件复位退出掉电模式。中断的开始清零 PD 位，退出掉电模式。保持外部中断端口为低重新起动振荡器,在回到高电平来完全退出之前,信号至少保持 1024 个时钟周期的低电平。在退出中断服务程序后，被中断程序紧接着从引起掉电模式命令的后一条指令重新开始执行。硬件复位同上电复位一样都能启动器件。

为了正确退出掉电模式，在 V_{DD} 恢复到它的正常工作电压之前,复位或外部中断将不应被执行。一定要保持 V_{DD} 在正常工作电压足够长的时间让振荡器振荡和稳定（推荐 30ms）。

1.10.5.3 Standby 模式(Stop clock)

Standby 同掉电模式相似，所不同的是掉电模式是通过软件命令进入，而 Standby 模式是

通过外部硬件关闭器件的外部时钟进入。在 Standby 模式下，片内 SRAM 和 SFR 的数据被保持。外部恢复供应时钟时，器件从下一指令继续工作。

表 9-10-2 概括了不同的省电模式，包括进入和退出过程和 MCU 的功能。

表 9-10-2： 省电模式

模式	实现方式	MCU 状态	退出方式
空闲模式	软件(置位 PCON 的 IDL 位)	时钟在运行。中断、串行口和定时/计数器是激活的，PC 停止。在 Idle 模式下，ALE 和 PSEN#处于高电平。所有寄存器保持不变。	中断或硬件复位。中断开始，清零 IDL，退出 Idle 模式，在中断服务程序（ISR）RETI 指令后，程序从进入 Idle 模式的后一指令开始执行。用户可以考虑在进入 Idle 模式的指令后放两或三条 NOP 指令来避免可能出现的问题。硬件复位同上电复位一样都能启动器件。
掉电模式	软件（置 PCON 的 PD 位）	时钟停止。片内 SRAM 和 SFR 数据保持。在掉电模式下，ALE 和 PSEN#保持在低电平。对电平敏感的外部中断有效，如果允许的话。	对电平敏感的外部中断或硬件复位。中断开始，清零 PD 位，退出掉电模式，在中断服务程序（ISR） RETI 指令后，程序继续从进入掉电模式指令的后一指令开始执行。用户可以考虑在进入掉电模式的指令后放两或三条 NOP 指令来避免可能出现的问题。硬件复位同上电复位一样都能启动器件。
Standby 模式（时钟停止）	外部硬件门关闭到 MCU 的外部时钟，门应该与输入时钟转换同步(low-to-high 或 high-to-low）	时钟停止。保持片内 SRAM 和 SFR 的数据。ALE 和 PSEN#保持在时钟停止前的电平。	开启外部时钟。程序继续从关闭时钟指令的下一条指令开始执行。

1.10.6 时钟输入选择

图 9-9-2 显示了内部反相放大器的输入和输出（XTAL1，XTAL2），反相放大器用于片内振荡器。当通过外部时钟源驱动器件时，XTAL2 浮空不接，XTAL1 接外部时钟源。在启动时，外部振荡器可能要承受在 XTAL1 上的高电容性负载，由于放大器和它的反馈电容的交互作用。然而，一旦外部信号满足 V_{IL} 和 V_{IH} 的要求时，电容就不要超过 15 pF。

1.10.7 推荐的电容值（对晶体振荡器）

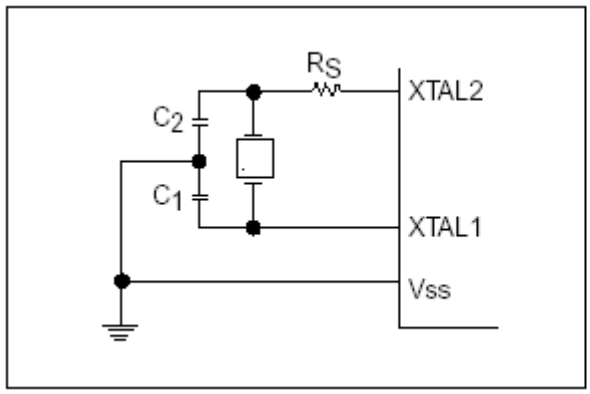
晶体生产厂家、提供的电源和其它因素可能导致电路性能不一样。对不同设计，C1 和

C2 的值应作适当调整。下表列出了在给定频率下 C1 和 C2 的典型值。如果外部元件条件满足，电路依然超载,那就要加上一个串行电阻（RS）。

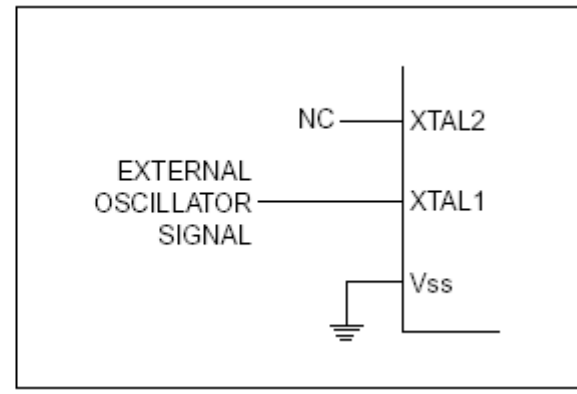
推荐的值（对晶体振荡器）

频率	C1 和 C2	RS
<8MHz	90-110pF	100Ω
8-12MHz	18-22pF	200Ω
>12MHz	18-22pF	200Ω

关于片内振荡器的更多详细信息在《FlashFlex51 振荡器电路设计应用说明》中可找到。



使用片内振荡器



外部时钟启动

图 9-9-2：振荡器特性

1.11 电气规格

绝对最大承受值（如果超过下表所列的“绝对最大承受值”可能对器件造成永久性的损害。将器件暴露在绝对最大承受条件下可能影响器件的可靠性。）

操作温度……-55℃~+125℃

保存温度……-65℃~+150℃

EA#到 V_{SS} 电压……-0.5V~+14.0V

其它引脚到 V_{SS} 的瞬时电压(<20ns)……-1.0V~+6.5V

P1.5、P1.6、P1.7 的最大输出电流……20mA

其它引脚最大输出电流……15 mA

包装功耗能力（Ta=25℃）……1.5W

过孔线焊接温度（10 秒）……300℃

表面装配线焊接温度……240℃

输出缩减电流……50mA

1．输出缩短不超过一秒。在同一时刻只能有一个缩减输出.(基于包装散热局限性,器件

没有能量损耗)

注意：此规范包含正在生产中的一些新产品的信息。此规范如没有特别说明不得更改。

1.11.1 工作范围

表 9-11-1 工作范围

符号	描述	最大值	最小值	单位
Ta	在中心值下的环境温度范围			
	标准	0	+70	℃
	工业	-40	+85	℃
V_{DD}	提供电压	2．7	5．5	V
f_{OSC}	振荡器频率	0	40	MHz
	IAP 模式下振荡器频率	0．25	40	MHz

1.11.2 可靠性

表 9-11-2 ：可靠性

符号	参数	最小值	单位	测试方法
N_{END}[1]	持久性	10000	周期	JEDEC 标准 A117
T_{DR}[1]	数据保持期限	100	年	JEDEC 标准 A103
I_{LTH}[1]	Latch up	100+I_{DD}	mA	JEDEC 标准 78

1．此参数在最初条件下测定，设计或处理可能影响这个参数。

1.11.3 DC 电气特性

表 9-10-3 DC 电气特性:40MHz;4.5-5.5V

Tamb=0 ℃ ~+70 ℃ 或 -40 ℃ ~ +85 ℃ ， 40MHz 驱 动 ； 4.5-5.5V ； V_{SS}=0V

Symbol	Parameter	Test Conditions	Min	Max	Units
V_{IL}	Input Low Voltage	4.5 < V_{DD} < 5.5	-0.5	0.2V_{DD} - 0.1	V
V_{IH}	Input High Voltage	4.5 < V_{DD} < 5.5	0.2V_{DD} + 0.9	V_{DD} + 0.5	V
V_{IH1}	Input High Voltage (XTAL1, RST)	4.5 < V_{DD} < 5.5	0.7V_{DD}	V_{DD} + 0.5	V
V_{OL}	Output Low Voltage (Ports 1.5, 1.6, 1.7)	V_{DD} = 4.5V			
		I_{OL} = 16mA		1.0	V
V_{OL}	Output Low Voltage (Ports 1, 2, 3)[1]	V_{DD} = 4.5V			
		I_{OL} = 100μA[2]		0.3	V
		I_{OL} = 1.6mA[2]		0.45	V
		I_{OL} = 3.5mA[2]		1.0	V
V_{OL1}	Output Low Voltage (Port 0, ALE, PSEN#)[1,3]	V_{DD} = 4.5V			
		I_{OL} = 200μA[2]		0.3	V
		I_{OL} = 3.2mA[2]		0.45	V
V_{OH}	Output High Voltage (Ports 1, 2, 3, ALE, PSEN#)[4]	V_{DD} = 4.5V			
		I_{OH} = -10μA	V_{DD} - 0.3		V
		I_{OH} = -30μA	V_{DD} - 0.7		V
		I_{OH} = -60μA	V_{DD} - 1.5		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)[4]	V_{DD} = 4.5V			
		I_{OH} = -200μA	V_{DD} - 0.3		V
		I_{OH} = -3.2mA	V_{DD} - 0.7		V
V_{BOD}	Brown-out Detection Voltage		3.85	4.15	V
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3)	V_{IN} = 0.4V	-1	-75	μA
I_{TL}	Logical 1-to-0 Transition Current (Ports 1, 2, 3)[5]	V_{IN} = 2V		-650	μA
I_{LI}	Input Leakage Current (Port 0)	0.45 < V_{IN} < V_{DD}-0.3		±10	μA
R_{RST}	RST Pulldown Resistor		40	225	kΩ
C_{IO}	Pin Capacitance[6]	@ 1 MHz, 25°C		15	pF
I_{DD}	Power Supply Current[7]				
	In-Application Mode				
	@ 20 MHz			70	mA
	@ 40 MHz			88	mA
	Active Mode				
	@ 20 MHz			25	mA
	@ 40 MHz			45	mA
	Idle Mode				
	@ 20 MHz			9.5	mA
	@ 40 MHz			20	mA
	Standby (Stop Clock) Mode	T_{amb} = 0°C to +70°C		100	μA
		T_{amb} = -40°C to +85°C		125	μA
	Power Down Mode	Minimum V_{DD} = 2V			
		T_{amb} = 0°C to +70°C		40	μA
		T_{amb} = -40°C to +85°C		50	μA

表 9-11-4：DC 电路特性:40MHz;2.7-3.6V

Tamb=0 ℃ ～ +70 或 -40 ℃ ～ +85 ℃ ， 25MHz ； 2.7-3.6V ； V_{SS}=0V

Symbol	Parameter	Test Conditions	Min	Max	Units
V_{IL}	Input Low Voltage	$2.7 < V_{DD} < 3.3$	-0.5	0.7	V
V_{IH}	Input High Voltage	$2.7 < V_{DD} < 3.3$	$0.2V_{DD} + 0.9$	$V_{DD} + 0.5$	V
V_{IH1}	Input High Voltage (XTAL1, RST)	$2.7 < V_{DD} < 3.3$	$0.7V_{DD}$	$V_{DD} + 0.5$	V
V_{OL}	Output Low Voltage (Ports 1.5, 1.6, 1.7)	V_{DD} = 2.7V			
		I_{OL} = 16mA		1.0	V
V_{OL}	Output Low Voltage (Ports 1, 2, 3)[1]	V_{DD} = 2.7V			
		I_{OL} = 100µA[2]		0.3	V
		I_{OL} = 1.6mA[2]		0.45	V
		I_{OL} = 3.5mA[2]		1.0	V
V_{OL1}	Output Low Voltage (Port 0, ALE, PSEN#)[1,3]	V_{DD} = 2.7V			
		I_{OL} = 200µA[2]		0.3	V
		I_{OL} = 3.2mA[2]		0.45	V
V_{OH}	Output High Voltage (Ports 1, 2, 3, ALE, PSEN#)[4]	V_{DD} = 2.7V			
		I_{OH} = -10µA	V_{DD} - 0.3		V
		I_{OH} = -30µA	V_{DD} - 0.7		V
		I_{OH} = -60µA	V_{DD} - 1.5		V
V_{OH1}	Output High Voltage (Port 0 in External Bus Mode)[4]	V_{DD} = 2.7V			
		I_{OH} = -200µA	V_{DD} - 0.3		V
		I_{OH} = -3.2mA	V_{DD} - 0.7		V
V_{BCD}	Brown-out Detection Voltage		2.25	2.55	V
I_{IL}	Logical 0 Input Current (Ports 1, 2, 3)	V_{IN} = 0.4V	-1	-75	µA
I_{TL}	Logical 1-to-0 Transition Current (Ports 1, 2, 3)[5]	V_{IN} = 2V		-650	µA
I_{LI}	Input Leakage Current (Port 0)	$0.45 < V_{IN} < V_{DD}$-0.3		±10	µA
R_{RST}	RST Pulldown Resistor			225	kΩ
C_{IO}	Pin Capacitance[6]	@ 1 MHz, 25°C		15	pF
I_{DD}	Power Supply Current[7]				
	In-Application Mode			70	mA
	Active Mode			22	mA
	Idle Mode			6.5	mA
	Standby (Stop Clock) Mode	T_{amb} = 0°C to +70°C		70	µA
		T_{amb} = -40°C to +85°C		88	µA
	Power Down Mode	Minimum V_{DD} = 2V			
		T_{amb} = 0°C to +70°C		40	µA
		T_{amb} = -40°C to +85°C		50	µA

1．在稳定状态下，I_{OL} 必须限定在如下所示范围内：

经过端口的最大 I_{OL} ：15mA， 每个 8 位口的最大 I_{OL} ：2 mA,所有输出的最大 I_{OL} ：71mA, 如果 I_{OL} 超过了测试值，V_{OH} 可能超过相关值。引脚不能降低电流到低于表中所列测试值。

2．电容加在 P0 和 P2 上可能引起各种干扰，这些干扰可能与 ALE 和 P1、P3 的 V_{OLS} 重叠。在对总线操作期间当引脚状态由 1 到 0 的转换时，干扰将导致外部电容放电进入 P0、P2。在最坏情况下（电容放电>100Pf）,ALE 上的干扰脉冲可能超过 0.8V。在这种情况下，最好做法是在 ALE 端接一触发器，或者用一带一触发器的地址锁存器起滤波作用。

3．在 P1、ALE、PSEN#上接入一值为 100pF 的电容，所有其它输入端接入一值为 80pF 的电容。

4． 当地址位稳定后，加在 P0、P2 上的电容可能引起 ALE 和 PSEN#上的电压即刻下降为 V_{DD}-0.7

5．当 P1、P2、P3 由外部驱动从 1 变为 0 时，将产生一个转换电流。当 V_{IN} 达到 2V 时，转换电流将达到它的最大值。

6．引脚电容量是表现出来的而不是测量出来的。EA#最大是 25pF。

7．参考图 9-10-1、9-10-2、9-10-3 和 9-10-4 中的测试环境。在掉电状态下 V_{DD} 最小为 2.0V。

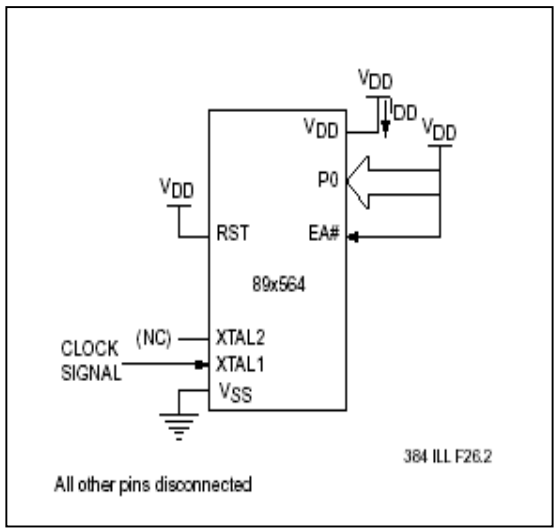


图 9-11-1: I_{DD} 测试条件:ACTIVE 模式

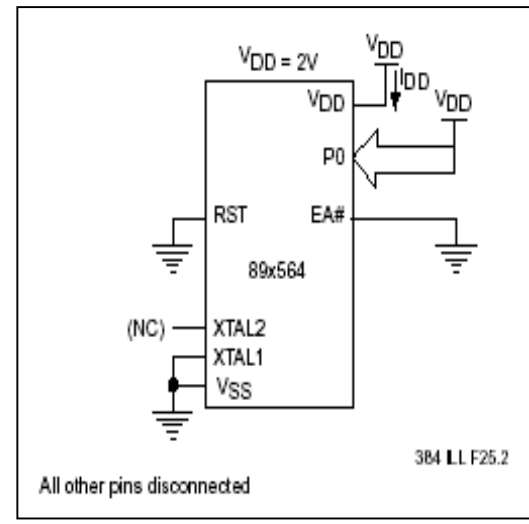


图 9-11-2: I_{DD} 测试条件:Power_Down 模式

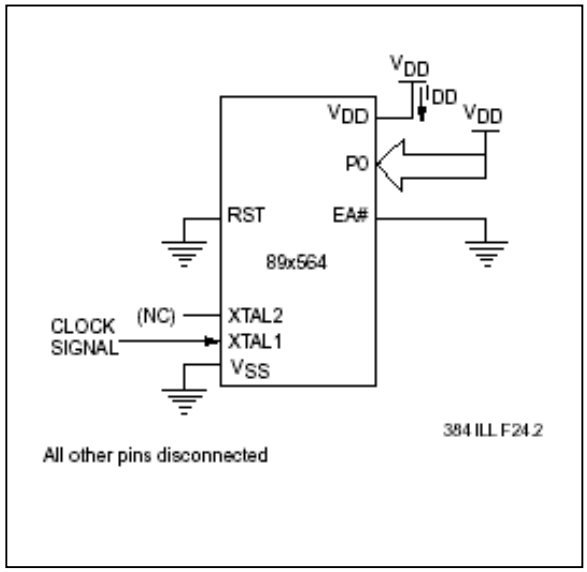


图 9-10-3 I_{DD} 测试条件:IDLE 模式

1.11.4 AC 电气特性

AC 特性（在下列条件下：在 P0、ALE、PSEN#上接入一值为 100pF 的电容，所有其它输出端接上一值为 80pF 的电容）

表 9-11-5：AC 电气特性

Tamb=0℃～+70℃或-40℃+85℃，V_{DD}=2.7～3.6@25MHz,4.5～5.5@40MHz,V_{SS}=0V。

Symbol	Parameter	Oscillator						Units
		25MHz		40MHz		Variable		
		Min	Max	Min	Max	Min	Max	
$1/T_{CLCL}$	Oscillator Frequency					0	40	MHz
T_{LHLL}	ALE Pulse Width	65		35		$2T_{CLCL}$ - 15		ns
T_{AVLL}	Address Valid to ALE Low	15				T_{CLCL} - 25 (3V)		ns
				10		T_{CLCL} - 15 (5V)		ns
T_{LLAX}	Address Hold After ALE Low	15				T_{CLCL} - 25 (3V)		ns
				10		T_{CLCL} - 15 (5V)		ns
T_{LLIV}	ALE Low to Valid Instr In		95				$4T_{CLCL}$ - 65 (3V)	ns
					55		$4T_{CLCL}$ - 45 (5V)	ns
T_{LLPL}	ALE Low to PSEN# Low	15				T_{CLCL} - 25 (3V)		ns
				10		T_{CLCL} - 15 (5V)		ns
T_{PLPH}	PSEN# Pulse Width	95				$3T_{CLCL}$ - 25 (3V)		ns
				60		$3T_{CLCL}$ - 15 (5V)		
T_{PLIV}	PSEN# Low to Valid Instr In		65				$3T_{CLCL}$ - 55 (3V)	ns
					25		$3T_{CLCL}$ - 50 (5V)	ns
T_{PXIX}	Input Instr Hold After PSEN#					0		ns
T_{PXIZ}	Input Instr Float After PSEN#		35				T_{CLCL} - 5 (3V)	ns
					10		T_{CLCL} - 15 (5V)	ns
T_{AVIV}	Address to Valid Instr In		120				$5T_{CLCL}$ - 80 (3V)	ns
					65		$5T_{CLCL}$ - 60 (5V)	ns
T_{PLAZ}	PSEN# Low to Address Float		10		10		10	ns
T_{RLRH}	RD# Pulse Width	200				$6T_{CLCL}$ - 40 (3V)		ns
				120		$6T_{CLCL}$ - 30 (5V)		
T_{WLWH}	Write Pulse Width (WE#)	200				$6T_{CLCL}$ - 40 (3V)		ns
				120		$6T_{CLCL}$ - 30 (5V)		
T_{RLDV}	RD# Low to Valid Data In		110				$5T_{CLCL}$ - 90 (3V)	ns
					75		$5T_{CLCL}$ - 50 (5V)	ns
T_{RHDX}	Data Hold After RD#	0		0		0		ns
T_{RHDZ}	Data Float After RD#		55				$2T_{CLCL}$ - 25 (3V)	ns
					38		$2T_{CLCL}$ - 12 (5V)	ns
T_{LLDV}	ALE Low to Valid Data In		230				$8T_{CLCL}$ - 90 (3V)	ns
					150		$8T_{CLCL}$ - 50 (5V)	ns
T_{AVDV}	Address to Valid Data In		270				$9T_{CLCL}$ - 90 (3V)	ns
					150		$9T_{CLCL}$ - 75 (5V)	ns
T_{LLWL}	ALE Low to RD# or WR# Low	95	145			$3T_{CLCL}$ - 25 (3V)	$3T_{CLCL}$ + 25 (3V)	ns
				60	90	$3T_{CLCL}$ - 15 (5V)	$3T_{CLCL}$ + 15 (5V)	
T_{AVWL}	Address to RD# or WR# Low	85				$4T_{CLCL}$ - 75 (3V)		ns
				70		$4T_{CLCL}$ - 30 (5V)		ns
T_{QVWX}	Data Valid to WR# High to Low Transition		0		0		0	ns
T_{WHQX}	Data Hold After WR#	13				T_{CLCL} - 27 (3V)		ns
				5		T_{CLCL} - 20 (5V)		ns

表 9-11-5：AC 电气特性

Tamb=0℃～+70℃或-40℃+85℃，V_{DD}=2.7～3.6@25MHz,4.5～5.5@40MHz,V_{SS}=0V。

Symbol	Parameter	Oscillator						Units
		25MHz		40MHz		Variable		
		Min	Max	Min	Max	Min	Max	
T_{QVWH}	Data Valid to WR# High	433				$7T_{CLCL}$ - 70 (3V)		ns
				125		$7T_{CLCL}$ - 50 (5V)		ns
T_{RLAZ}	RD# Low to Address Float		0		0		0	ns
T_{WHLH}	RD# to WR# High to ALE High	43	123			T_{CLCL} - 25 (3V)	T_{CLCL} + 25 (3V)	ns
				10	40	T_{CLCL} - 15 (5V)	T_{CLCL} + 15 (5V)	ns

1.11.5 AC 特性

符号说明 每个时序符号有 5 个特征，第一特征是一个‘T’（ 代表时间）。其它特征根据所在位置代

表信号名称或信号的逻辑状态。下面列出了所有的特征及它们所代表的意思。

A：地址 Q：输出数据 C：时钟 R：RD#信号 D：输入数据

T：时间 H：逻辑高 V：有效 I：指令 W：WR#信号

L：逻辑低或 ALE X：不是有效逻辑电平 P：PSEN# Z： 高阻抗

例如：T_{AVLL}=从地址有效到 ALE 为低的时间

T_{LLPL}=从 ALE 低到 PSEN#为低的时间

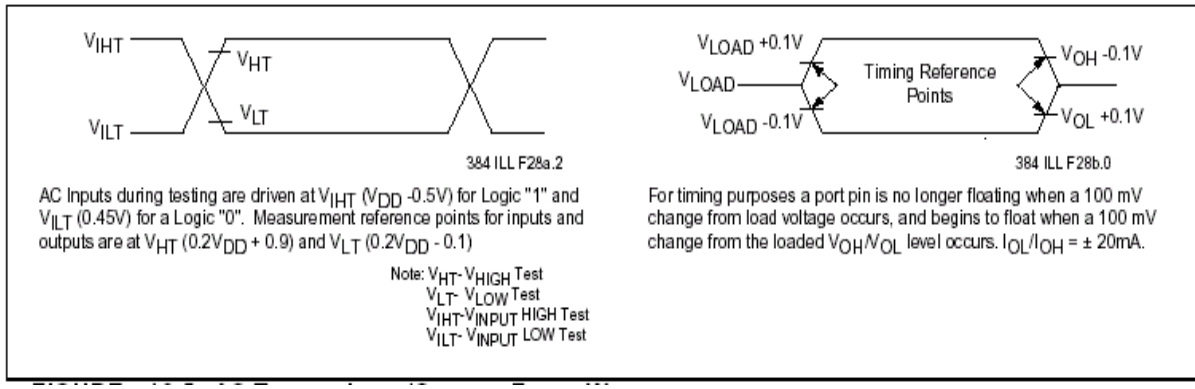


图 9-11-5 AC 测试输入/输出 ，飘移波形

1.11.6 FLASH 存储器时序图

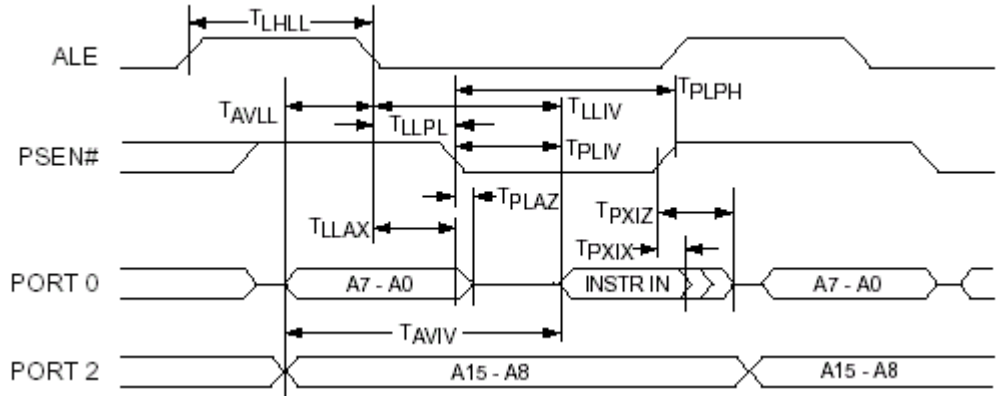


图 9-11-6 外部程序存储器读周期

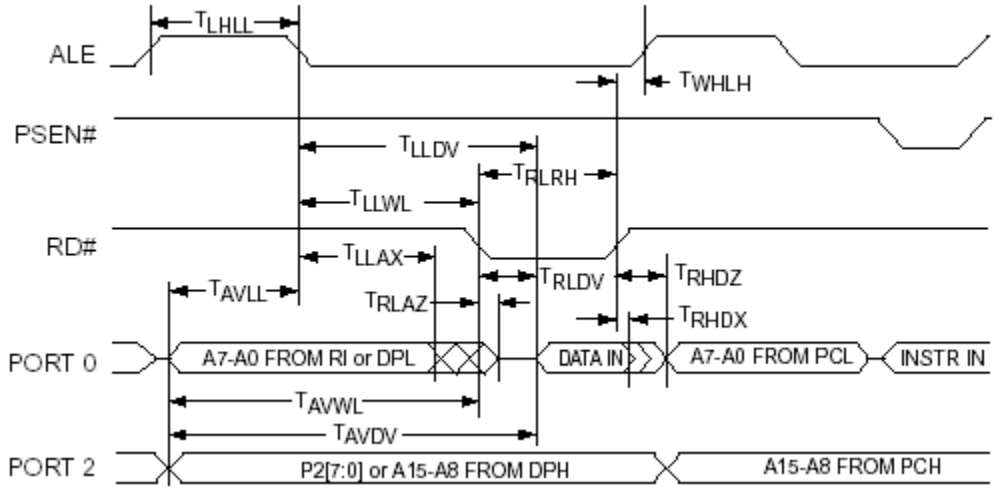


图 9-11-7 外部数据存储器读周期

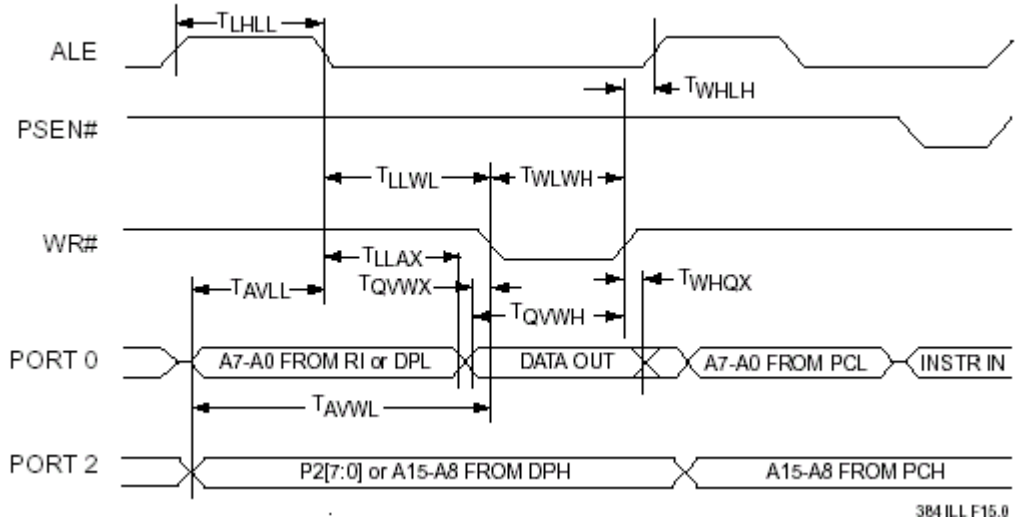


图 9-11-8：外部数据存储器写周期

表 9-10-6：外部时钟驱动

Symbol	Parameter	Oscillator						Units
		25MHz		40MHz		Variable		
		Min	Max	Min	Max	Min	Max	
$1/T_{CLCL}$	Oscillator Frequency					0	40	MHz
T_{CHCX}	High Time					$0.35T_{CLCL}$	$0.65T_{CLCL}$	ns
T_{CLCX}	Low Time					$0.35T_{CLCL}$	$0.65T_{CLCL}$	ns
T_{CLCH}	Rise Time		20		10			ns
T_{CHCL}	Fall Time		20		10			ns

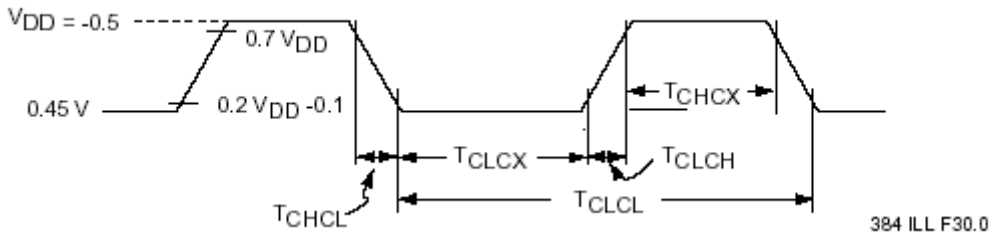


图 9-11-9:外部时钟驱动波形图

表 9-11-7:串行口时序

Symbol	Parameter	Oscillator						Units
		25MHz		40MHz		Variable		
		Min	Max	Min	Max	Min	Max	
T_{XLXL}	Serial Port Clock Cycle Time	0		0.36		$12T_{CLCL}$		ms
T_{QVXH}	Output Data Setup to Clock Rising Edge	700		117		$10T_{CLCL}$ - 133		ns
T_{XHQX}	Output Data Hold After Clock Rising Edge	50				$2T_{CLCL}$ - 117		ns
				0		$2T_{CLCL}$ - 50		ns
T_{XHDX}	Input Data Hold After Clock Rising Edge	0		0		0		ns
T_{XHDV}	Clock Rising Edge to Input Data Valid		700		117		$10T_{CLCL}$ - 133	ns

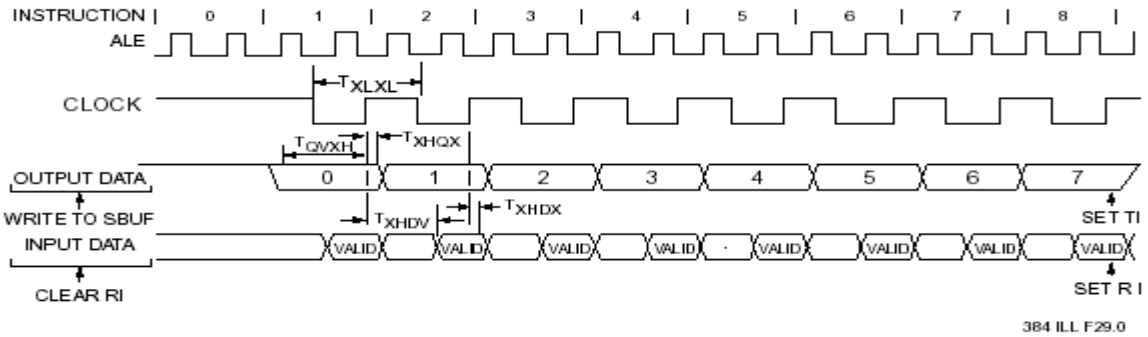
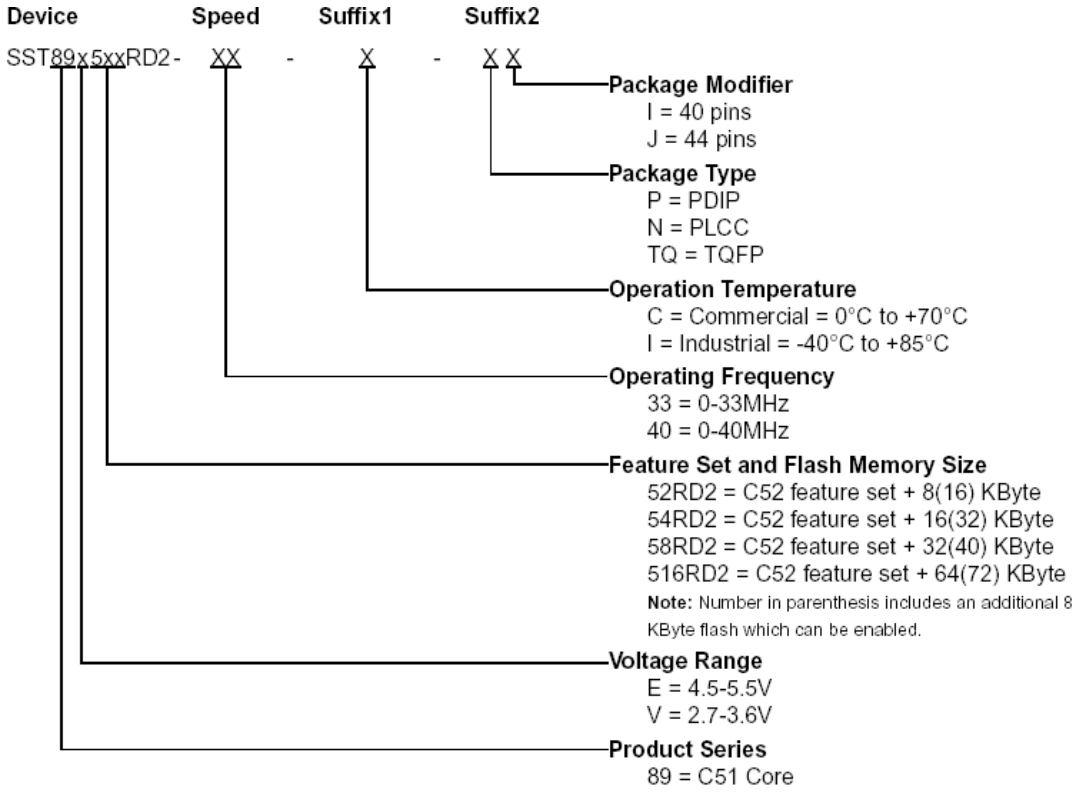
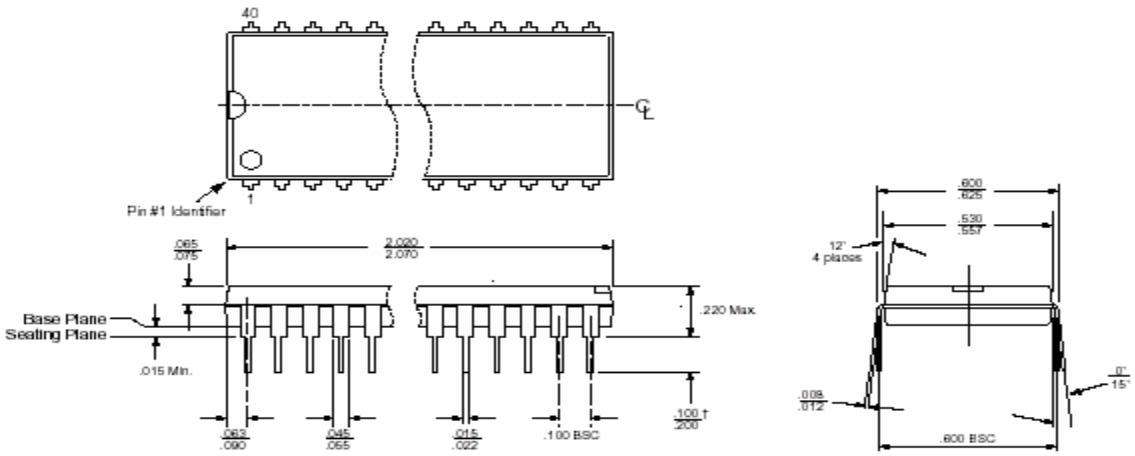


图 9-11-10 Ship Register Mode Timing Waveforms(移位寄存器模式时序图)

1.12 产品命名规则



1.13 封装图



PDIP-40 SST 封装代码:PI

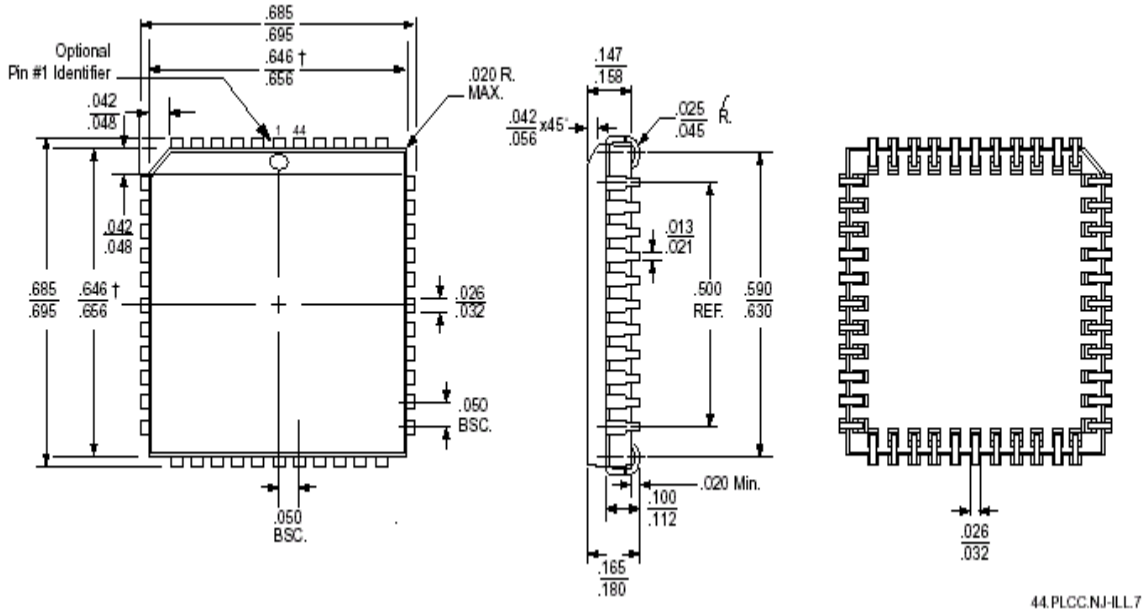
说明 1. 尽管很多尺寸要求很严格，仍与 JEDEC 发布的 95 MS-011 AC 尺寸相符合（除非特别说明）。

+= JEDEC 最小是 0.115，SST 精确度比它要高。

2．所有线性尺寸都是英寸（min/max）。

3．封装尺寸不包括毛边，允许最大毛边为 0.010 英寸。

TOP VIEW SIDE VIEW BOTTOM VIEW



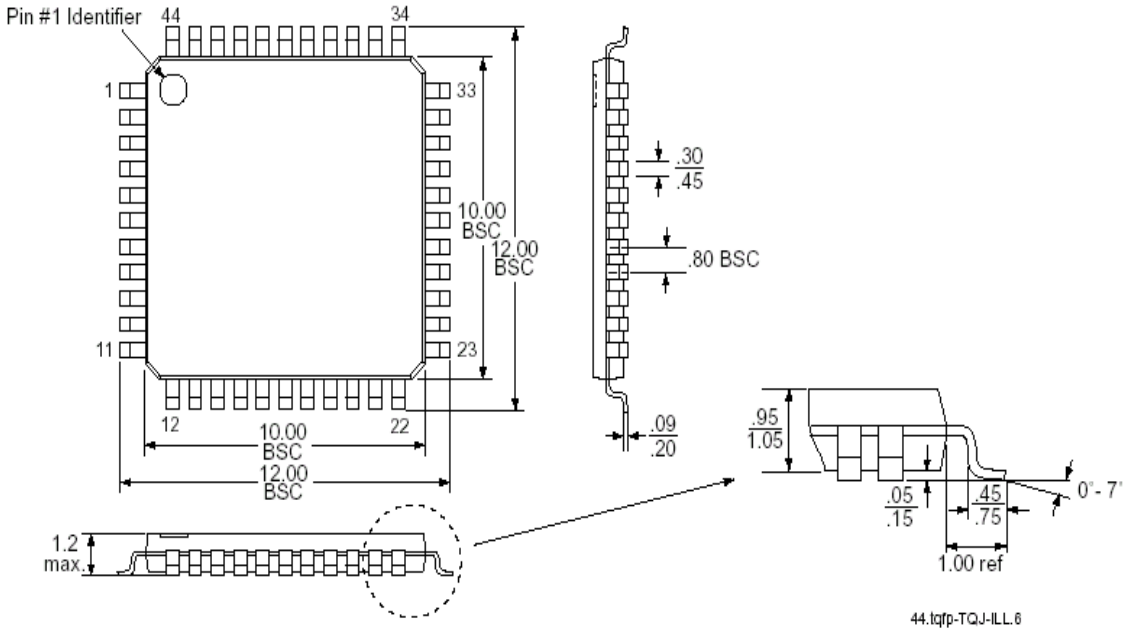
PLCC-44 封装代码:NJ

1． 尽管很多尺寸要求很严格，仍与 JEDEC 发布的 95 MS-018 AC 尺寸相符合。
+=JEDEC 最小是 0.650，SST 精确度比它高。

2．所有线性尺寸都是英寸（min/max）.

3．封装尺寸不包括毛边，允许最大毛边为 0.008 英寸.

4．误差:4mils



TQFP-44 封装代码:TQJ

1. 尽管很多尺寸要求很严格，仍与 JEDEC 发布的 95 MS-026 ACB 尺寸相符合。
2. 所有线性尺寸都是毫米（min/max）.
3. 误差:0.1(±0.05) 毫米.
4. 封装尺寸不包括毛边，允许最大毛边为 0.25mm.

香港弘微科技有限公司

SST 单片机中文教程

第 一 章 绪 论

单片机的英文名是 Single Chip Microcontroller，可以译为单片微控制器。目前也有 SingleChip Microcomputer 的叫法，译为单片微型计算机，还有一种叫法是英文 MicroControlUnit，简称 MCU，所以以后大家见到上面这些字眼，应该知道都是指的单片机。单片机也是计算机，它和我们所常见的 PC 机有些不同，它是将 CPU、存储器、输入/输出等功能都集成在一片芯片上的微型计算机，所以我们的早期研究工作者称它为单片机，并且一直沿用至今。

单片机的应用十分广泛，使用数量大得惊人，几乎是 PC 机数量的成百甚至上千倍。常见的电话、彩电、冰箱、热水器、微波炉，大的火箭、太空船、卫星、导弹等等，无处不能找到单片机的踪影，只是由于它不喜宣扬、默默无闻、辛勤工作所以我们一直未能感觉到它的存在而已。

1946 年，自第一台计算机问世以来，计算机就一直朝着两个方向发展：一是具有海量数据计算能力的数据处理计算机，如 PC 机；一是具有强大控制功能的控制计算机，如 MCU，MCU 广泛应用于控制领域。MCU 常以嵌入到对方内部的方式工作，我们把以这种方式工作的处理器叫做嵌入式处理器，由嵌入式处理器组成的系统就叫做嵌入式处理系统（EmbededSystem）。

我们以前接触过的 PC 机是基于冯若依曼的体系结构，而我们今天接触的单片机和我们今后要接触的 DSP 等都是基于的哈佛体系结构。哈佛结构和冯若依曼结构有一定的区别，我们将在后面章节详细分析。

第一片单片机发布于 1971 年，是 Intel 公司的 Intel 4004,并且配备有随机存取存储器 RAM，只读存储器 ROM 和移位寄存器等芯片，构成第一台 MCS－4 微型计算机。1972 年 4 月 Intel 公司又发布了 Intel 8008,这也就是单片机发展的第一阶段。

单片机的第二阶段以 MCS－48 为代表（1974～1978 年）。这一系列的单片机内集成有 8 位 CPU、并行 I/O 口、8 位定时器/计数器，寻址范围不大于 4K，且无串行口。

第三阶段的单片机是高性能的单片机（1978～1983 年）。这一阶段的单片机普遍带有串行口，有多级中断处理系统、16 位定时器/计数器。片内 RAM、ROM 容量加大，寻址范围达 64K，有的片内还带 A/D 转换接口。这类单片机有 Intel 公司的 MCS－51,Motorola 公司的 6801 和 Zilog 公司的 Z80。前几年这类单片机是主流产品，广泛应用于我国工业各个领域。

目前单片机正朝着功能完善化，专一化方向发展，以满足各个不同用户的需求。

由于最近几年芯片的设计制造成本大大降低，导致单片机的发展日新月异，估计今后几年会有以下发展趋势：

1、 高可靠性

由于在许多场合，机器对系统稳定性有严格的要求，一次故障将会造成严重事故或重大损失。这就要求单片机系统具有高度的稳定性，虽然可以通过许多方式提高系统稳定性，但单片机的可靠性是首要的。

2、 高集成度

高集成度不但可以提高单片机的抗干扰能力，而且还能降低整个系统的成本。目前由于生产成本的降低，许多厂家开始把原先属于单片机的外围设备都集成到芯片的里面了。譬如 A/D 接口，FlashROM，看门狗等都集成到了芯片内部。例如 SST89C54 里面就集成了看门狗 WDT。

3、 更快的处理速度

为了完成实时处理功能，在高速系统中，单片机需要有快的反应速度。这就要求单片机具有更快的处理能力，譬如有的单片机为了提高处理速度竟在里面集成了硬件乘法器，一般单片机的乘法功能是由加法器完成的。

总之，任何事物的发展都是顺应时代的发展的要求的，单片机也不例外，用户对单片机提出什么样的要求，单片机就会朝着什么样的方向发展。

第 二 章 8051 单片机的结构和原理

在前一章我们介绍了单片机的起源，发展及应用，从这一章开始我们将以 8051 核为例详细介绍单片机的硬件、软件、接口和应用。

2.1 51 系列单片机的结构

51 单片机最初是由 Intel 公司开发设计的，但后来 Intel 公司把 51 核的设计方案卖给了几家大的电子设计生产商，譬如 SST、Philip、Atmel 等大公司。如是市面上出现了各式各样的但均以 51 为内核的单片机，倒是 Intel 公司自己的单片机却显得逊色了。这些各大电子生产商推出的单片机都兼容 51 指令、并在 51 的基础上扩展一些功能而内部结构是与 51 一致的，在前一章我们已经提到 51 单片机在今后很长一段时间内仍是主流，所以我们的教材将还是以 51 核为例给大家进行详细的介绍。

2.1.1 51 系列单片机的结构框图

我们假设读者是已经学完了计算机的组成原理，所以下面出现的有关计算机的专有名词就不做详细介绍了。

我们知道我们 PC 机的 CPU 是基于冯诺伊曼的体系结构，然而 MCU（单片机）、Dsp（数字信号处理器）都是基于哈佛结构的体系结构。哈佛结构与冯诺伊曼结构有很大的不同，在冯诺伊曼体系结构下只有一个地址空间，ROM 和 RAM 可以随意安排在这一地址范围内的不同空间，即 ROM 和 RAM 地址统一分配。CPU 访问存储器时，一个地址对应唯一的存储单元，可能是 ROM，也可能是 RAM。而哈佛结构下 ROM 和 RAM 是分开编址，即程序和数据分开保存，访问时用不同的指令加以区分，并可同时访问，在这样的体系结构下有利于提高指令的执行速度。在后面的章节我们将详细介绍单片机的存储器配置。

图 2－1 所示为 MCS－51 系列单片机的基本结构框图。

从结构框图我们可以看出在这一小块芯片上，集成了一个微型计算机的各个组成部分。这些部分包括：

（1） 一个 8 位的微处理器（CPU）。

（2） 片内数据存储器 RAM（128B / 256B），用以存放可以读 / 写的数据，如运算的中间结果、最终结果以及欲显示的数据等，SST89 系列单片机最多提供 1K 的 RAM。

（3） 片内程序存储器 ROM / EPROM（4KB / 8KB），用以存放程序、一些原始数据和表格。但也有一些单片机内部不带 ROM / EPROM，如 8031，8032，80C31 等。目前单片机的发展趋势是将 RAM 和 ROM 都集成在单片机里面，这样既方便了用户进行设计又提高了系统的抗干扰性。SST 公司推出的 89 系列单片机分别集成了 16K、32K、64K Flash 存储器，可供用户根据需要选用，读者可查看书的后面部分。

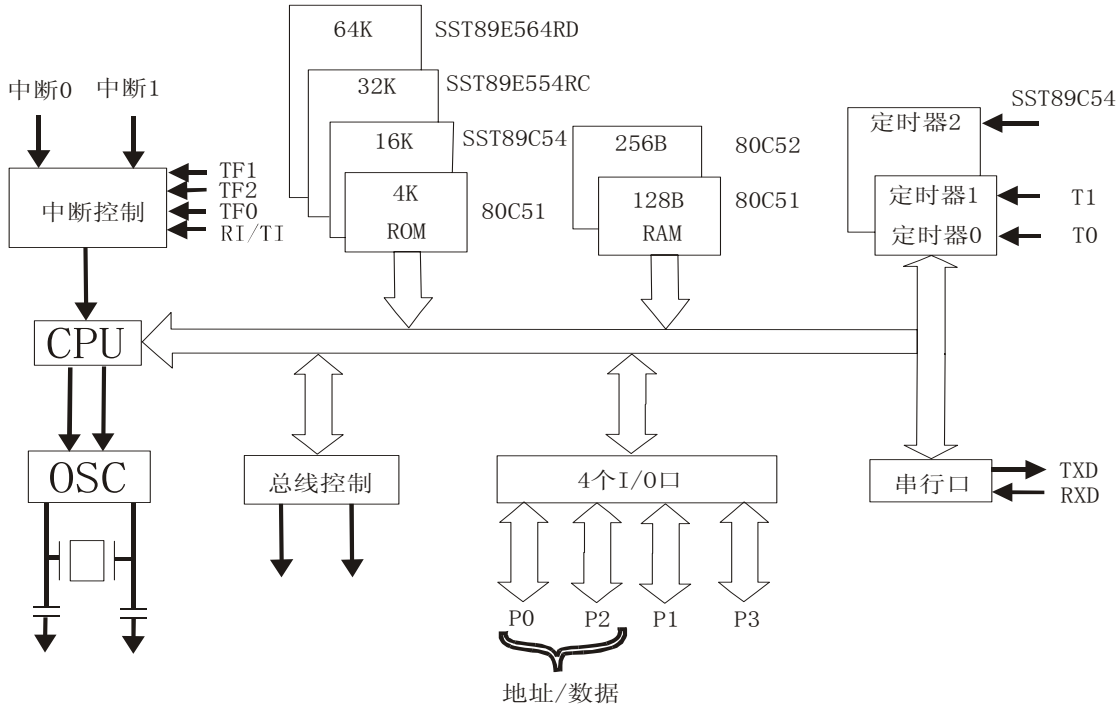


图 2－1　51 系列单片机的基本结构框图

（4）　四个 8 位并行 I / O 接口 P0~P3，每个口既可以用作输入，也可以用作输出。

（5）　两个定时器 / 计数器，每个定时器 / 计数器都可以设置成计数方式，用以对外部事件进行计数，也可以设置成定时方式，并可以根据计数或定时的结果实现计算机控制。为方便设计串行通信，目前的 52 系列单片机都会提供 3 个 16 位定时器/计数器。

（6）　五个中断源的中断控制系统。现在新推出的单片机都不只 5 个中断源，例如 SST89E58RD 就有 9 个中断源。

（7）　一个全双工 UART（通用异步接收发送器）的串行 I / O 口，用于实现单片机之间或单机与微机之间的串行通信。

（8）　片内振荡器和时钟产生电路，但石英晶体和微调电容需要外接。最高允许振荡频率为 12MHz。SST89V58RD 最高允许振荡频率达 40MHz，因而大大的提高了指令的执行速度。

以上各个部分通过内部数据总线相互连接。

早期的 51 系列单片机有十多个品种，目前已发展到数百种，我们可以看看早期单片机的性能如表 2－1 所示，拿它和现代新型单片机比较，我们会发现它们的性能相差很大，可以参见后文的 SST89 系列单片机性能。

表 2-1 51 系列单片机性能表

ROM 形式			片 内 ROM/B	片 内 RAM/B	寻址范围	I / O			中断源
片内 ROM	片 内 EPROM	外 接 EPROM				计数器	并行口	串行口	
8051	8751	8031	4K	128	2×64K	2×16b	4×8b	1	5
80C51	87C51	80C31	4K	128	2×64K	2×16b	4×8b	1	5
8052	8752	8032	8K	256	2×64K	3×16b	4×8b	1	6
80C552	87C52	80C32	8K	256	2×64K	3×16b	6×8b	2	15

表 2－1 中，单片机型号带“C”表示所用的是 CMOS 工艺，具有功耗低的优点。如 8051 的功耗为 630mW，而 80C51 的功耗只有 120mW，它用于低功耗的便携式产品或航天技术领域中。

MCS－51 系列单片机的湿度适用范围也较微处理器芯片 Z80，8080 等宽，其温度范围为：

民品（商业用） 0℃ ～ 70℃
工业品 －40℃ ～ 85℃
军用品 －55℃ ～ 125℃

市场上的销售品多为工业品，其稳定性和抗干扰性都优于微处理器芯片。

***注 SST 系列单片机性能表**

型 号	最高时钟频率Hz		Flash 存储器	RAM	串口 UART	PCA	中断源	优先级	DPTR 数据指针	降低 EMI	掉电检测	看门狗	双倍速	P4 口	SPI
	5V /E	3V /V													
SST89E/V516RD2	40M	33M	64K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V58RD2	40M	33M	32K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V54RD2	40M	33M	16K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V52RD2	40M	33M	8K+8K	1KB	1ch+	5ch	8	4	2	✓	✓	✓	✓	✓	✓
SST89E/V54RC	33M	25M	16K+1K	512B	1ch+	0	8	4	2	✓	✓	✓	✓		✓
SST89E/V52RC	33M	25M	8K+1K	512B	1ch+	0	8	4	2	✓	✓	✓	✓		✓

2.1.2 51 单片机内部结构

8051 / 8751 / 8031 芯片的外部结构引脚和指令系统完全兼容，其内部结构除 ROM / EPROM 不同外，其余完全相同。现时流行的 52 单片机引脚也是与之兼容的。

8051 单片机内部结构如图 2-2 所示。一个完整的计算机应该由运算器、控制器、存储器（ROM 及 RAM）、数据总线和 I / O 接口组成。一般微处理器（如 8086）就只包括运算器和控制器两部分。和一般微处理器相比，8051 增加了四个 8 位 I / O 口、一个串行口、4KB ROM、128BRAM、很多工作寄存器及特殊功能寄存器（SFR），所以单片机具有比微处理器更强大的控制功能，单片机是专为进行控制设计的，而常见的微处理器是用于运算功能的，下图各部分的功能描述

一、中央处理单元（CPU）

和 PC 机的 CPU 一样，它是单片机的核心，是计算机的控制和指挥中心，由运算器和控制器等部件组成。

1. 运算器

运算器包括一个可进行 8 位算术运算和逻辑运算的单元 ALU，8 位的暂存器 1、暂存器 2，8 位的累加器 ACC，寄存器 B 和程序状态寄存器 PSW 等。

ALU：可对 4 位（半字节）、8 位（一字节）和 16 位（双字节）数据进行操作。能做加、减、乘、除、加 1、减 1、BCD 数十进制调整及比较等算术运算和与、或、异或、求补及循环移位等逻辑操作。

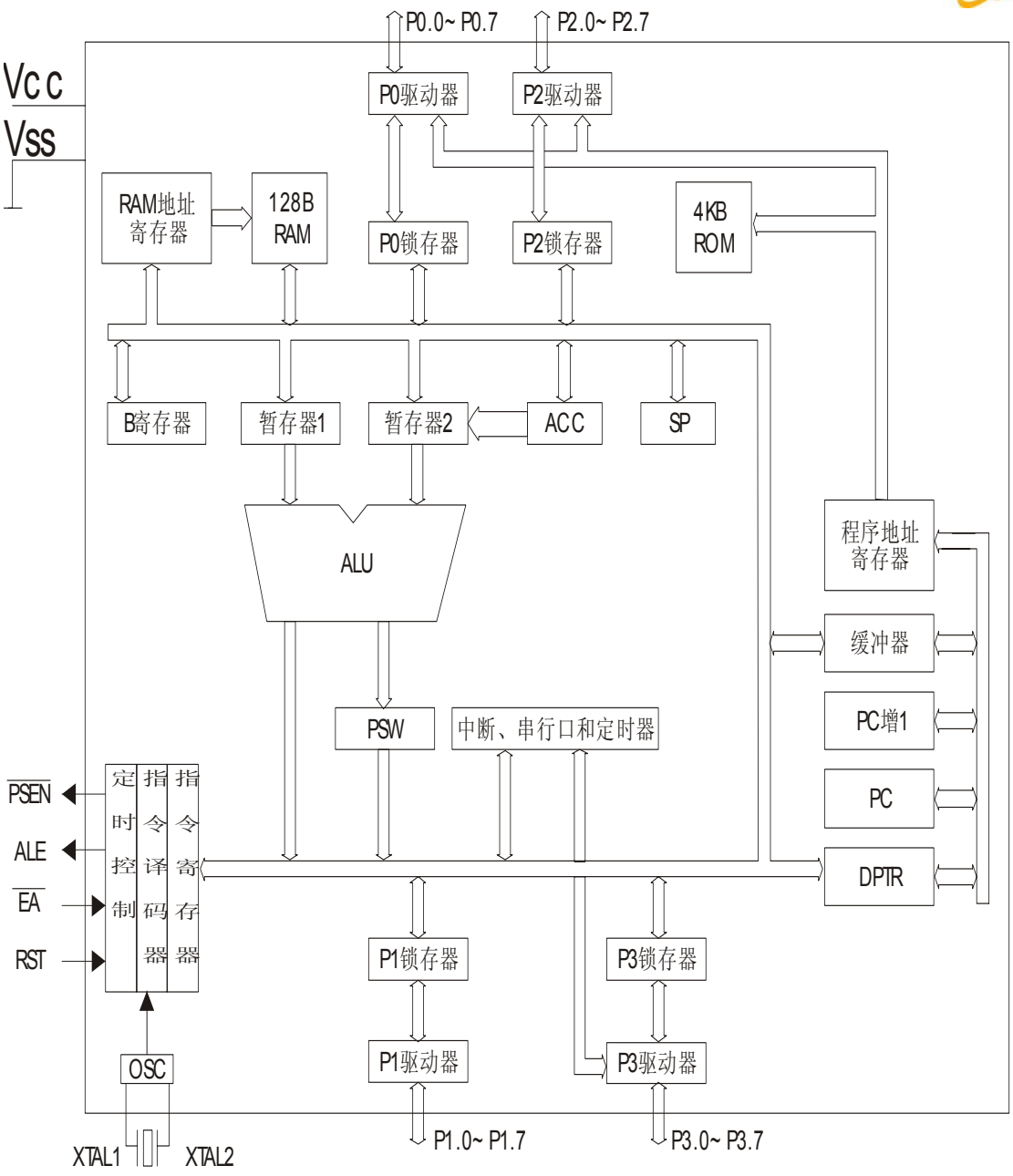


图 2-2　8051 单片机内部结构图

ACC：累加器 ACC 经常作为一个运算数经暂存器 2 进入 ALU 的输入端，与另一个来自暂存器 1 的运算数进行运算，运算结果又送回 ACC。除此之外，ACC 在 8051 内部经常作为数据传送的中转站。同一般微处理器一样，它是最繁忙的一定寄存器了。在指令中用助记符 A 来表示。

PSW：程序状态字寄存器，8 位，用于指示指令执行后的状态信息，相当于一般微处理器的标志寄存器。PSW 中各位状态供程序查询和判别用。详见特殊功能寄存器 SFR 中介绍。

B：8 位寄存器，在乘、除运算时，B 寄存器用来存放一个操作数，也用来存放运算后的一部分结果；若不做乘、除运算时，则可作为通用寄存器使用。

另外，8051 片内还有一个布尔处理器，它以 PSW 中的进位标志位 CY 为其累加器（在布尔处理器及其指令中以 C 代替 CY），专门用于处理位操作：可执行置位、位清 0、位取反、位等于 1 转移、位等于 0 转移、位等于 1 转移并清 0 以及位累加器 C 与其他可位寻址的空间之间进行信息传送等位操作，也能使 C 与其他可寻址位之间进行逻辑“与”、逻辑“或”操作，结果存放在进位标志位（位累加器）C 中。

2．控制器

控制器包括程序计数器 PC、指令寄存器 IR、指令译码器 ID、振荡器及定时电路等。

程序计数器 PC：由两个 8 位的计数器 PCH 及 PCL 组成，共 16 位。PC 实际上是程序的字节地址计数器，PC 中的内容是将要招待的下一条指令的地址。改变 PC 的内容就可改变程序执行的方向。PC 可对 64KB 的 ROM（程序存储器）直接寻址，也可对 8051 片外 RAM（数据存储器）寻址。

指令寄存器 IR 及指令译码器 ID，由 ID 对指令译码并送 PLA 产生一定序列的控制信号，以执行指令所规定的操作。例如，控制 ALU 的操作、在 8051 片内工作寄存器间传送数据，以及发出 ACC 与 I / O 口（P0~P3）或存储器之间通信的控制信号等等。

振荡器及定时电路：8051 单片机片内有振荡电路，只需外接石英晶体和频率微调电容（2 个 30pF 左右），其频率范围为 1.2MHz~12MHz。该脉冲信号就作为 8051 工作的基本节拍，即时间的最小单位。8051 同其他计算机一样，在基本节拍的控制下协调地工作，就像一个乐队按着指挥的节拍演奏一样。

二、存储器

8051 片内有 ROM（程序存储器，只能读）和 RAM（数据存储器，可读可写）两类，它们有各自独立的存储地址空间，与一般微机的存储器配置方式很不相同。

1、 程序存储器（ROM）

8051 及 8751 的片内程序存储器容量为 4KB，地址从 0000H 一开始，用于存放程序和表格常数。

2、 数据存储器（RAM）

8051 / 8751 / 8031 片内数据存储器均为 128B，地址为 00H~7FH，用于存放运算的中间结果、数据暂存以及数据缓冲等。

在这 128B 的 RAM 中，有 32 个字节单元可指定为工作寄存器，这同一般微处理器不同，8051 的片内 RAM 和工作寄存器排在一个队列里统一编址。

由图 2－2 可见,8051 单片机内部还有 SP,DPTR,PCON,…,IE,IP 等特殊功能寄存器，它们也同 128 字节 RAM 在一个队列里编址，地址为 80H~FFH。在这 128 字节 RAM 单元中有 21 个特殊功能寄存器(SFR),这些特殊功能寄存器还包括 P0~P3 口锁存器。

如何使用 RAM 中的 32 个工作寄存器和特殊功能寄存器，后面将详细介绍。

三、I/O 接口

8051 有四个 8 位并行接口，即 P0~P3。它们都是双向端口，每个端口各有 8 条 I/O 线，为可输入 / 输出。P0~P3 口四个锁存器同 RAM 统一编址，可以把 I/O 口当作一般特殊功能寄存器来寻址。

2.2 51 单片机的引脚及其功能

MCS-51 系列中各种芯片的引脚是互相兼容的，如 8051,8071 和 8031 均采用 40 脚双列直插封装(DIP)方式。当然，不同芯片之间引脚功能也略有差异。8051 单片机是高性能单片机，因为受到引脚数目的限制，所以有不少引脚具有第二功能，如图 2-3 所示。

各引脚功能简要说明如下：

1. 电源引脚 Vcc 和 Vss

Vcc(40 脚)：电源端，为＋5V。

Vss(20 脚)：接地端。

2. 时钟电路引脚 XTAL1 和 XTAL2

XTAL2（18 脚）：接外部晶体和微调电容的一端；在 8051 片内它是振荡电路反相放大器的输出端，振荡电路的频率就是晶体固有频率。若需采用外部时钟电路时，该引脚输入外部时钟脉冲。

要检查 8051/8031 的振荡电路是否正常工作，可用示波器查看 XTAL2 端是否有脉冲信号输出。

XTAL1(19 脚)：接外部晶体和微调电容的另一端；在片内它是振荡电路反相放大器的输入端。在采用外部时钟时，该引脚必须接地。

3. 控制信号引脚 RST,ALE,PSEN 和 EA

RST/V_{PD}(9 脚)：RST 是复位信号输入端，高电平有效。当此输入端保持两个机器周期(24 个时钟振荡周期)的高电平时，就可以完成复位操作。RST 引脚的第二功能是 $V_{PD,}$即备用电源的输入端。当主电源 Vcc 发生故障，降低到低电平规定值时，将＋5V 电源自动接入 RST 端，为 RAM 提供备用电源，以保证存储在 RAM 中的信息不丢失，从而合复位后能继续正常运行。

ALE/PROG(ADDRESS LATCH ENABLE/PROGRAMMING,30 脚)：地址锁存允许信号端。当 8051 上电正常工作后，ALE 引脚不断向外输出正脉冲信号，此频率为振荡器频率 f_{OSC} 的 1/6。CPU 访问片外存储器时，ALE 输出信号作为锁存低 8 位地址的控制信号。

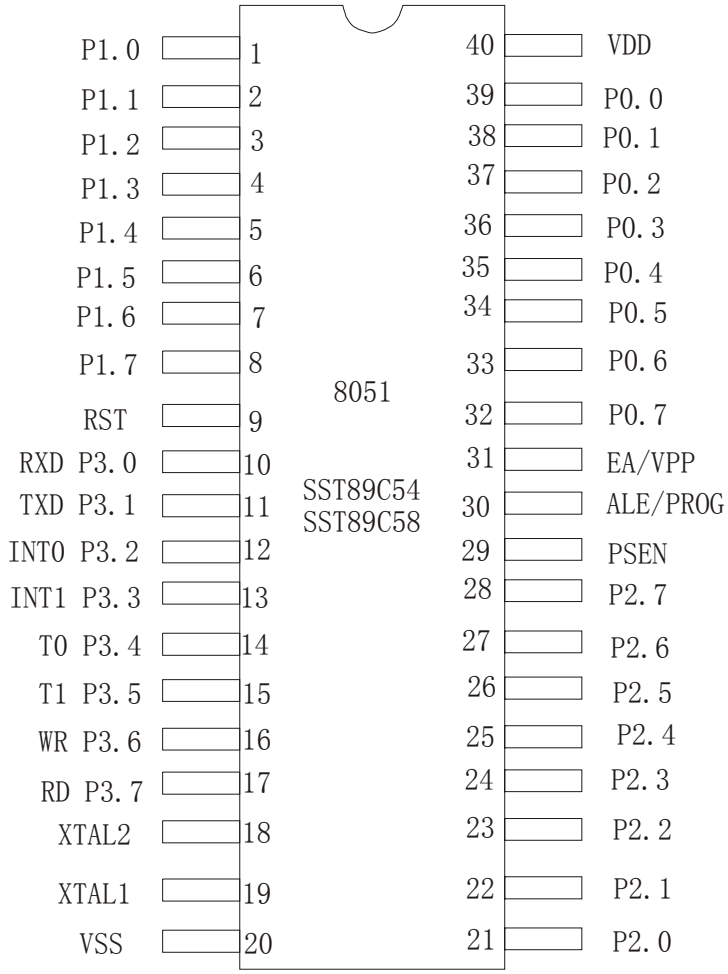


图 2-3　51 系列单片机引脚图

平时不访问片外存储器时，ALE 端也以振荡频率的 1/6 固定输出正脉冲，因而 ALE 信号可以用作对外输出时钟或定时信号。如果想确定 8051/8031 芯片的好坏，可用示波器查看 ALE 端是否有脉冲信号输出。如有脉冲信号输出，则 8051/8031 基本上是好的。

ALE 端的负载驱动能力为 8 个 LS 型 TTL(低功耗甚高速 TTL)负载。

此引脚的第二功能 PROG 在对片内带有 4KB EPROM 的 8751 编程写入(固化程序)时，作为编程脉冲输入端。

PSEN(PROGRAM STORE ENABLE,29 脚)：程序存储允许输出信号端。在访问片外程序存储器时，此端定时输出负脉冲作为读片外存储器的选通信号。此引肢接 EPROM 的 OE 端(见后面几章任何一个小系统硬件图)。PSEN 端有效，即允许读出 EPROM / ROM 中的指令码。

PSEN 端同样可驱动 8 个 LS 型 TTL 负载。

要检查一个 8051/8031 小系统上电后 CPU 能否正常到 EPROM / ROM 中读取指令码，也可用示波器看 PSEN 端有无脉冲输出。如有则说明基本上工作正常。

EA＃/Vpp(ENABLE ADDRESS/VOLTAGE PULSE OF PROGRAMING,31 脚)：外部程序存储器地址允许输入端/固化编程电压输入端。

当EA 引脚接高电平时，CPU 只访问片内 EPROM/ROM 并执行内部程序存储器中的指令，但当 PC（程序计数器）的值超过 0FFFH(对 8751/8051 为 4K)时，将自动转去执行片外程序存储器内的程序。

当输入信号 EA 引脚接低电平（接地）时，CPU 只访问外部 EPROM/ROM 并执行外部程序存储器中的指令，而不管是否有片内程序存储器。对于无片内 ROM 的 8031 或 8032,需外扩 EPROM，此时必须将 EA 引脚接地。

此引脚的第二功能是 Vpp 是对 8751 片内 EPROM 固化编程时，作为施加较高编程电压(一般 12V～21V）的输入端。

4.输入/输出端口 P0/P1/P2/P3

P0 口（P0.0～P0.7，39~32 脚)：P0 口是一个漏极开路的 8 位准双向 I/O 口。作为漏极开路的输出端口，每位能驱动 8 个 LS 型 TTL 负载。当 P0 口作为输入口使用时，应先向口锁存器（地址 80H）写入全 1,此时 P0 口的全部引脚浮空，可作为高阻抗输入。作输入口使用时要先写 1,这就是准双向口的含义。

在 CPU 访问片外存储器时，P0 口分时提供低 8 位地址和 8 位数据的复用总线。在此期间，P0 口内部上拉电阻有效。

P1 口（P1.0～P1.7，1~8 脚)：P1 口是一个带内部上拉电阻的 8 位准双向 I/O 口。P1 口每位能驱动 4 个 LS 型 TTL 负载。

在 P1 口作为输入口使用时，应先向 P1 口锁存地址（90H）写入全 1,此时 P1 口引脚由内部上拉电阻拉成高电平。

*** 注**：如果是 SST89 系列单片机，则它的 P1 口还有其它功能，现分列如下：

P1[0]	I/O	T2 ：定时器 /计数器 2 外部计数输入或时钟输出从定时器/计数器 2
P1[1]	I	T2EX： 定时器/计数器 2 捕捉/重装触发器和方向控制
P1[2]	I	EC1：PCA 定时器/计数器外部输入
P1[3]	I/O	CEX0：比较/捕捉外部输入输出模块，每个比较/捕捉模块连接到一 P1 口引脚，当不用于 PCA 时， 这个口用作标准 I/O
P1[4]	I/O	SS#： 主机输入、从机输出（SPI）或 CEX1：比较/捕捉外部输入输出模块
P1[5]	I/O	MOSI：主机输出，从机输入（SPI）或 CEX2：比较/捕捉外部输入输出模块
P1[6]	I/O	MISO：主机输入，从机输出（SPI）或 CEX3：比较/捕捉外部输入输出模块
P1[7]	I/O	SCK： 主机时钟输出、从机时钟输入或 CEX4：比较/捕捉外部输入输出模块

P2 口（P2.0～P2.7，21~28 脚)：P2 口是一个带内部上拉电阻的 8 位准双向 I/O 口。P2 口每位能驱动 4 个 LS 型 TTL 负载。

在访问片外 EPROM/RAM 时，它输出高 8 位地址。

P3 口（P3.0～P3.7，10~17 脚）：P3 口是一个带内部上拉电阻的 8 位准双向 I/O 口。P3 口每位能驱动 4 个 LS 型 TTL 负载。

P3 口与其它 I/O 端口有很大的区别，它的每个引脚都有第二功能。见下表：

P3[0]	I	RXD ：串行数据接收
P3[1]	O	TXD： 串行数据发送
P3[2]	I	INT0#：外部中断 0 输入
P3[3]	I	INT1#：外部中断 1 输入
P3[4]	I	T0：定时/计数器 0 的外部计数输入
P3[5]	I	T1：定时/计数器 1 的外部计数输入
P3[6]	O	WR#：外部数据存储器写选通
P3[7]	O	RD#：外部数据存储器读选通

2.3 8051 的存储器组织

2.3.1 程序存储器（Program Memory）和数据存储器（Data Memory）

所有的 51 系列单片机都有独立的程序存储器地址空间和数据存储器地址空间，见图 2-4

程序和数据存储器逻辑上的独立使得数据存储器能够被 8 位地址访问，这样 8 位的 CPU 能更快的保存和处理数据。此外，通过数据指针 DPTR 能产生 16 位的地址。程序存储器只能读出，不能写，访问空间可达 64K。芯片内部可提供 4K、8K、16K、32K 和 64K ROM，也可能没有，譬如 8031 不带 ROM，而 SST89E516RD 带 64KROM。在不带 ROM 和有外扩 ROM 的系统中，PSEN 是提供访问信号。数据存储器占用独立的地址空间，通过扩展访问可达 64K 外部 RAM。在需要访问外部数据 RAM 时，CPU 产生 RD 和 WR 信号。将 RD 和 PSEN 相与后，它的输出信号可以作为程序存储器和数据存储器的选通信号。

2.3.2 程序存储器

图 2-5 给出了程序存储器的低地址段的分布，CPU 复位后，程序从 0000H 地址开始执行。由图 2-5 可以看出，每个中断被分配一个地址空间，8 字节，每个中断有一个入口地址。当发生中断时，CPU 从相应的中断入口地址开始执行程序。例如发生外部中断 0,CPU 将从 0003H 处开始执行中断程序。

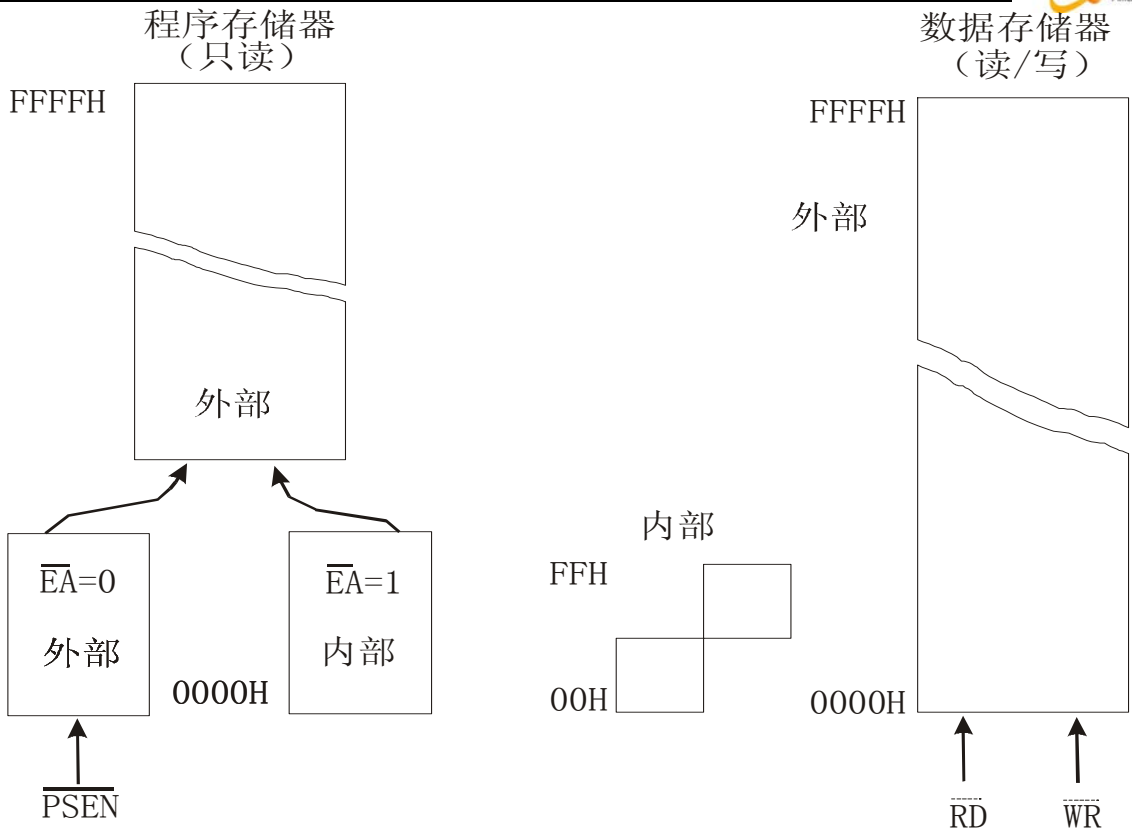


图 2-4　8051 存储器组织

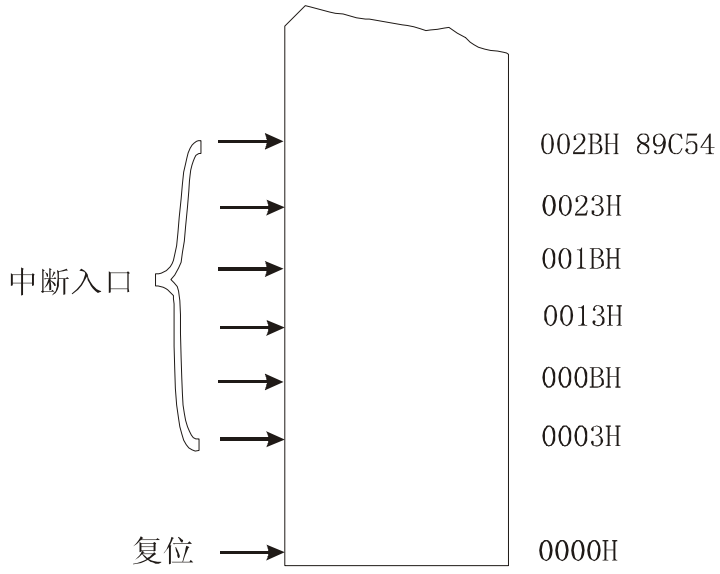


图 2-5　程序存储器的低地址段的分布

每个中断被分配 8 字节空间，如果中断服务程序足够的小能放入此空间，就不必再使用

跳转指令 JMP 到其它的地方去执行中断程序了。

对于 51 单片机，低地址 4K 的程序存储器既可使用芯片内部，也可使用片外扩展的 EPROM。这是通过将引脚 EA 拉高或接地来实现的。如果 EA 接 VCC，则从 0000H～0FFFH，程序从片内取指，而 1000H～FFFFH 程序将从片外取指。

对于 89C52 来说，则从 0000H～1FFFH，程序从片内取指，而 2000H～FFFFH 程序将从片外取指。

如果 EA 接地，那么所有的指令都将从片外程序存储器取得。PSEN 是外部存储器读选通信号。访问片外程序存储器的硬件结构图见图 2-6。

在访问片外程序存储器期间，这 16 根 I/O 口线被作为数据地址总线。其中 P0 为数据/地址复用总线。它送出外部程序存储器地址的低 8 位，然后等待外部程序存储器送来 8 位的代码字节。在这期间，地址锁定允许信号 ALE 发出脉冲，将地址低 8 位锁入地址锁存器。同时，P2 口送出高 8 位地址，然后 PSEN 发选通信号，这时程序代码被读入微控制器。

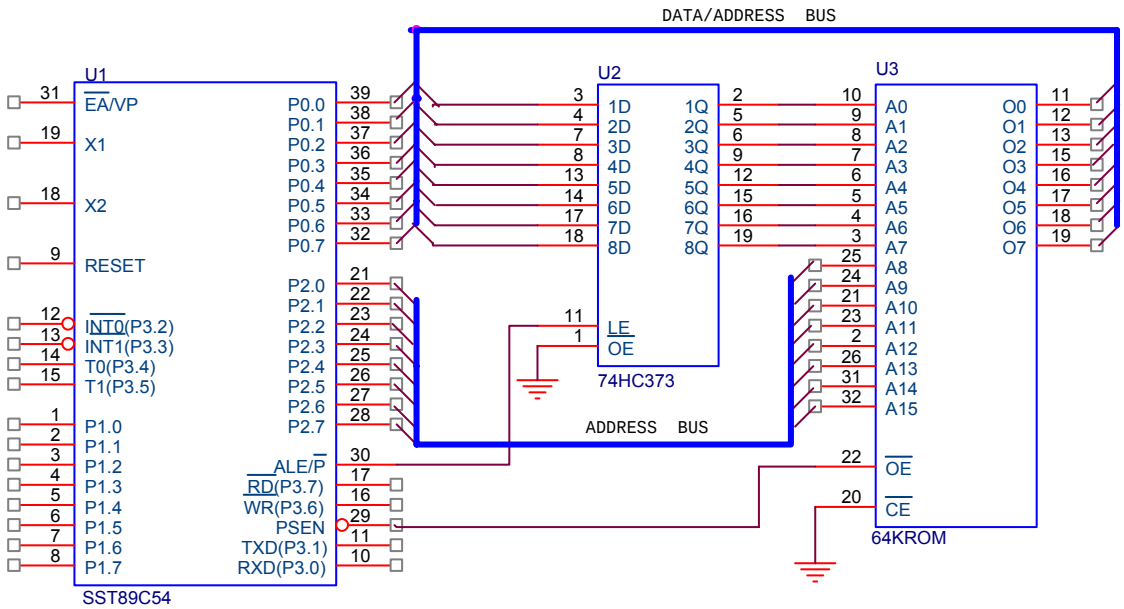


图 2-6 访问 64K 片外程序存储器

* 注：

SST89C58、SST89E58RD、SST89V58RD 片内带 32K ROM

SST89E516RD、SST89V516RD 片内带 64K ROM

这对于需要大容量 ROM 的系统来说既可以方便设计人员又可提高系统的稳定性。

2.3.3 数据存储器

图 2-4 的右半部分显示了内部和外部数据存储器的分布。图 2-7 给出访问 8K 外部数据存储器的结构图。

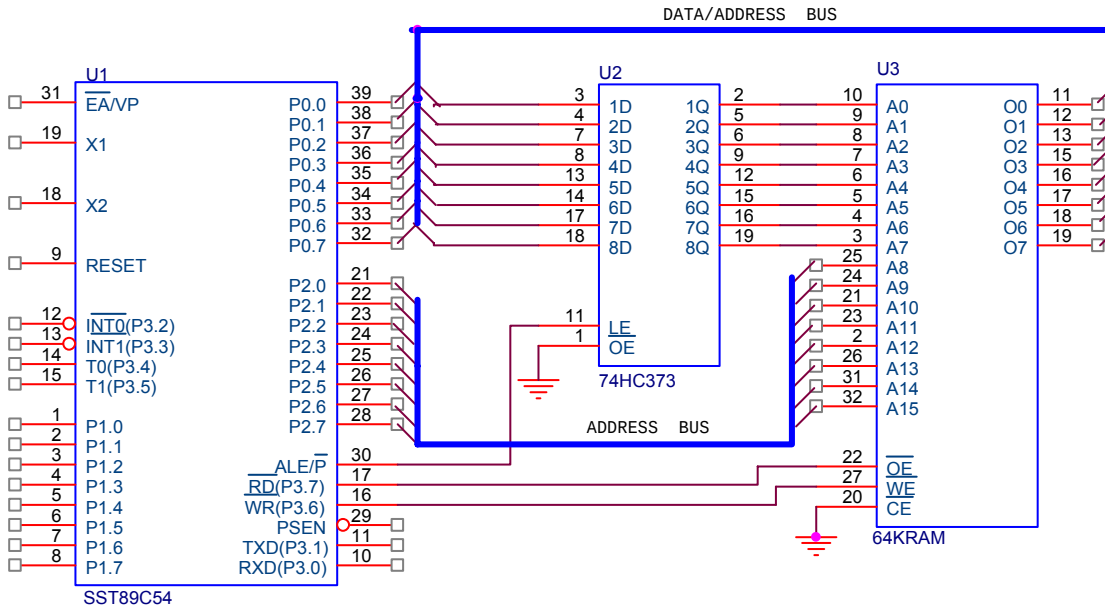


图 2-7 访问 64K 外部数据存储器

这类似如 CPU 访问外部程序存储器。P0 口仍然是数据地址口复用，而 P2 口的 3 根线用于 RAM 的页地址。在访问 RAM 时，CPU 产生 RD 和 WR 信号。

外扩 RAM 可达 64K，这是由于它有 16 根地址线，许多同学对此并不理解（2^{16}=64K）。

内部数据存储器分配图如图 2-8 所示：

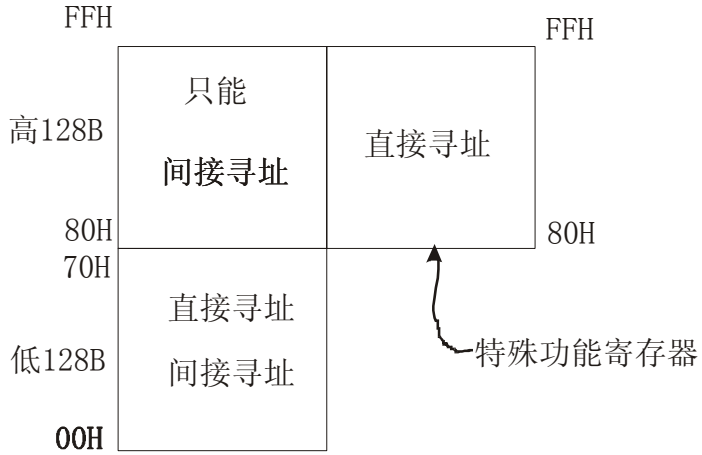


图 2-8 内部数据存储器分配

存储空间被分成 3 个快，通常是指低 128 字节，高 128 字节和 SFR（特殊功能寄存器）。内部数据存储器地址总是一字节宽，空间为 256 字节。然而，通过一个巧妙的方式，内

部 RAM 的这种地址模式却适用于 384 字节。高出 7FH 地址空间用直接访问方式和间接访问方式是访问的不同的地址空间，实际上一个是位地址，程序通过不同的指令来区分它们。

低 128 字节分布如图 2－9 所示。

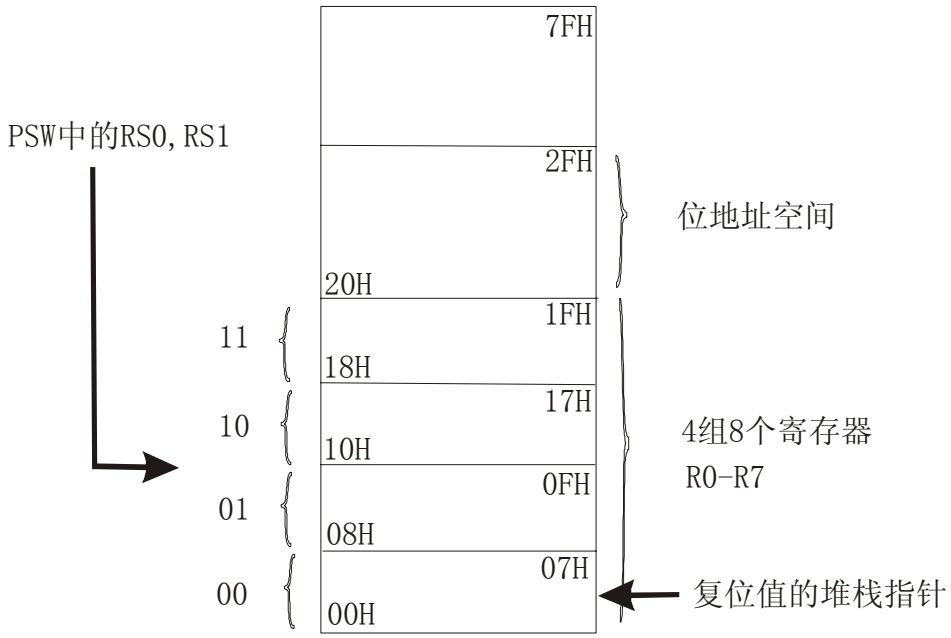


图 2-9 低 128 字节分布

最低的 32 字节被分成 4 块用于 8 个寄存器 R0～R7。通过设置 PSW 中的两位 RS0 和 RS1 来选择哪个寄存器快作为当前工作寄存器。由于寄存器指令要比直接访问指令短，所以使用寄存器操作能更有效的利用代码空间。接着寄存器块下来的 16 字节是位地址空间。51 单片机设置了位操作指令，这 128 位能按位地址进行操作。位地址从 00H 到 7FH。

所有的低 128 字节能被直接或间接寻址。高 128 字节只能间接寻址。

图 2-10 给出了一个特殊功能寄存器的简图。特殊功能寄存器也包括了 I/O 口、定时器、串口等等。通常，所有的 51 系列单片机都有相同的特殊功能寄存器，并分配了相同的地址空间。16 个特殊功能寄存器地址空间有的可进行位寻址，可位寻址的特殊功能寄存器的地址通常以 0,8 或 9 结尾。位地址在 80H～FFH 之间。

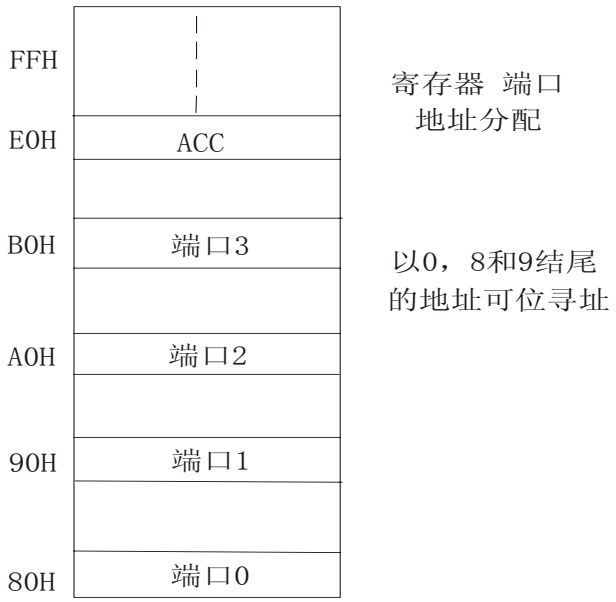


图 2-10 特殊功能寄存器简图

*** 注：**

SST89E58RD、SST89V58RD、SST89E516RD、SST89V516RD 都自带 1K RAM，这对于习惯用 C 语言编程的用户来说将会十分方便。

2.4 CPU 时序

所有 51 系列单片机都有一个片内振荡电路，用来为 CPU 提供工作时钟源。同时振荡电路工作需在 XTAL1 和 XTAL2 外接晶振，其连接示意图如图 2-11 所示。

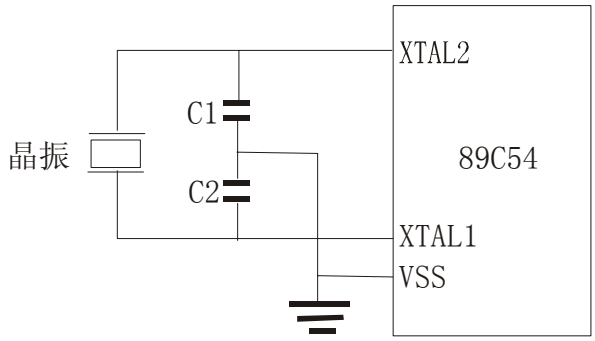


图 2-11 振荡电路

图 2-12 显示了如何使用外部时钟工作。

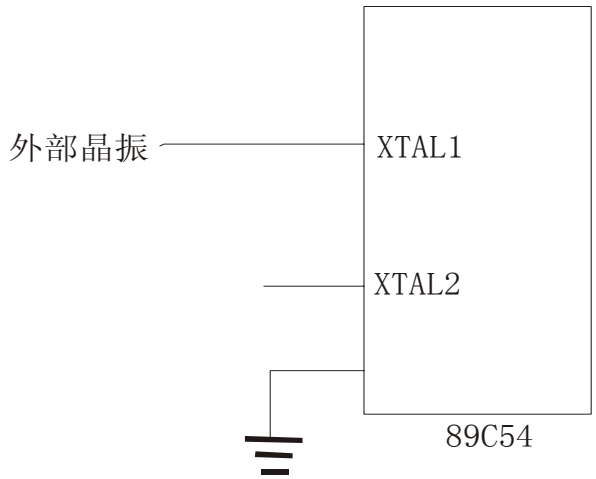


图 2-12 使用外部时钟

2.4.1 机器周期

一个机器周期由 6 个时序状态组成，S1 到 S6。每个状态持续 2 个晶振周期，因此一个机器周期包括 12 个晶振周期，如果晶振是 12MHz，那么一个机器周期是 1us。

每个状态被分为两部分，图 2-14 显示各种指令的取指和执行时序。图 2-13A 和 B 是单周期指令，在一个周期的状态 1（S1）时取操作码并锁入指令寄存器，第二次读操作数是在同

一周期的状态 4(S4)，在状态 6(S6)时指令执行结束。

MOVX 指令需两个机器周期，该指令没有读数据部分，图 2-13D 是该指令的执行时序。

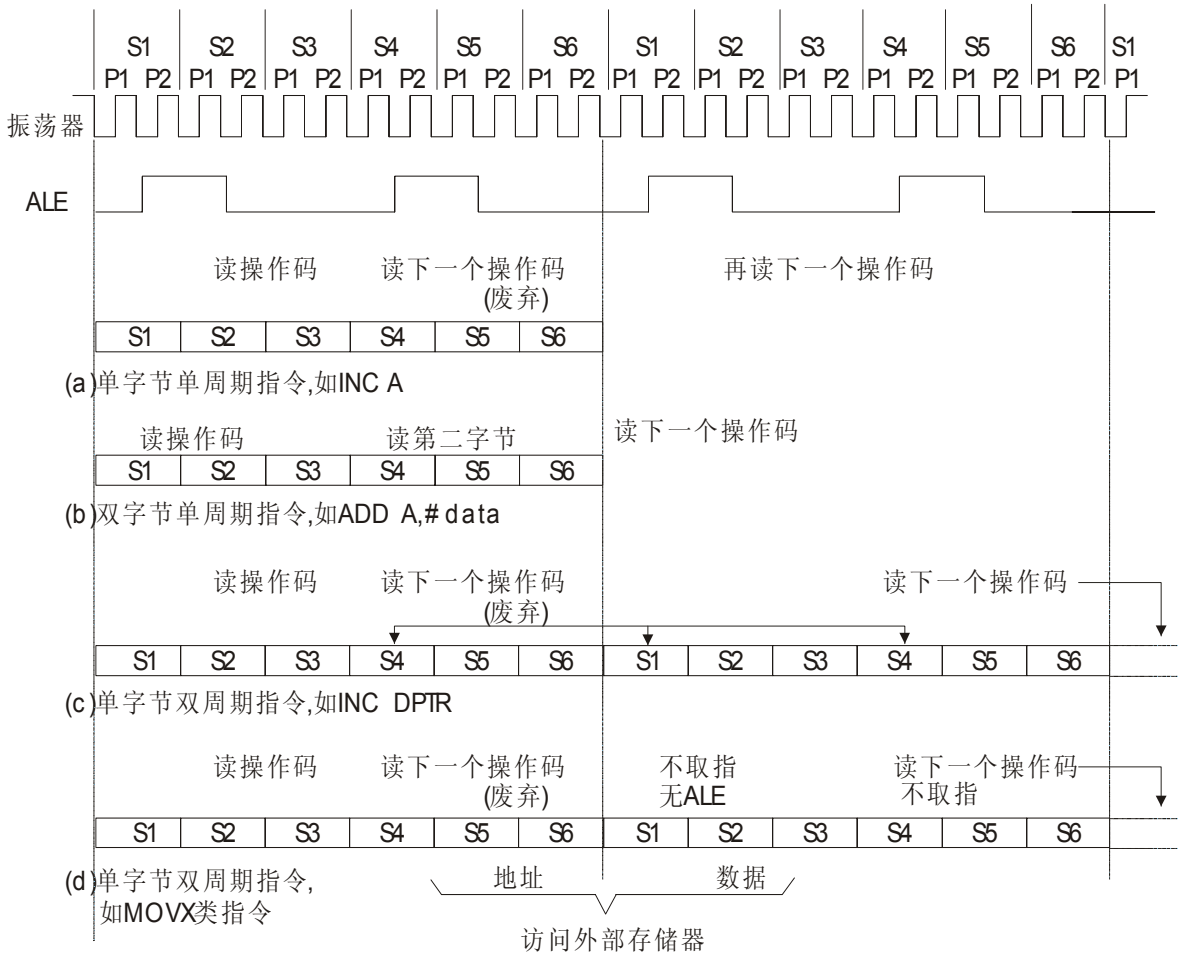


图 2-13 指令执行时序

程序存储器在片外还是片内对指令的执行是没有影响的。如果是外部程序存储器，那么 PSEN 每两个机器周期有效一次。当 CPU 执行内部程序存储器指令时，PSEN 无效。然而 ALE 持续发出有效信号，每机器周期两次，因此，可从 ALE 引脚获得固定频率的脉冲。

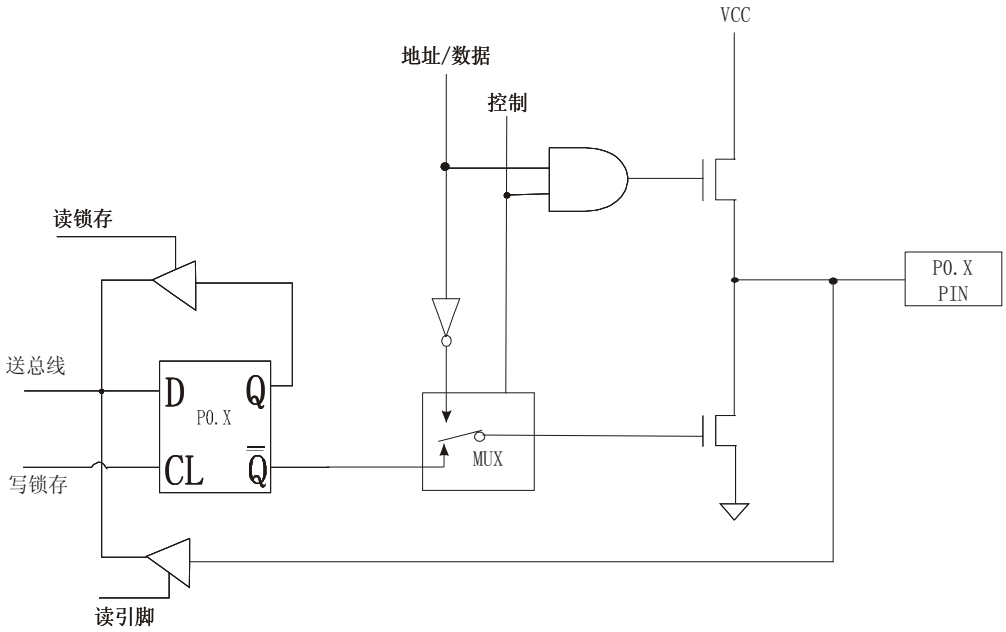
2.5 I/O 口结构和操作

8051 单片机的四个口都是双向 I/O 口，每个 I/O 口都由锁存器（特殊功能寄存器 P0～P3）、

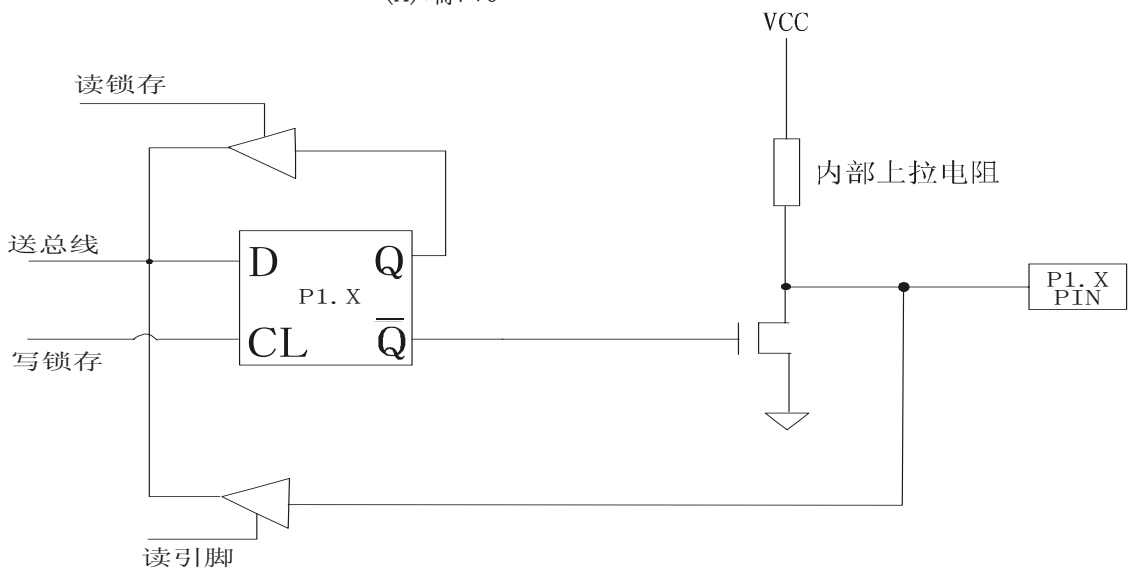
输出驱动和输入缓冲组成。P3 口的每个引脚还有第二功能，见 2.2 51 单片机的引脚及其功能。

2.5.1 I/O 口结构

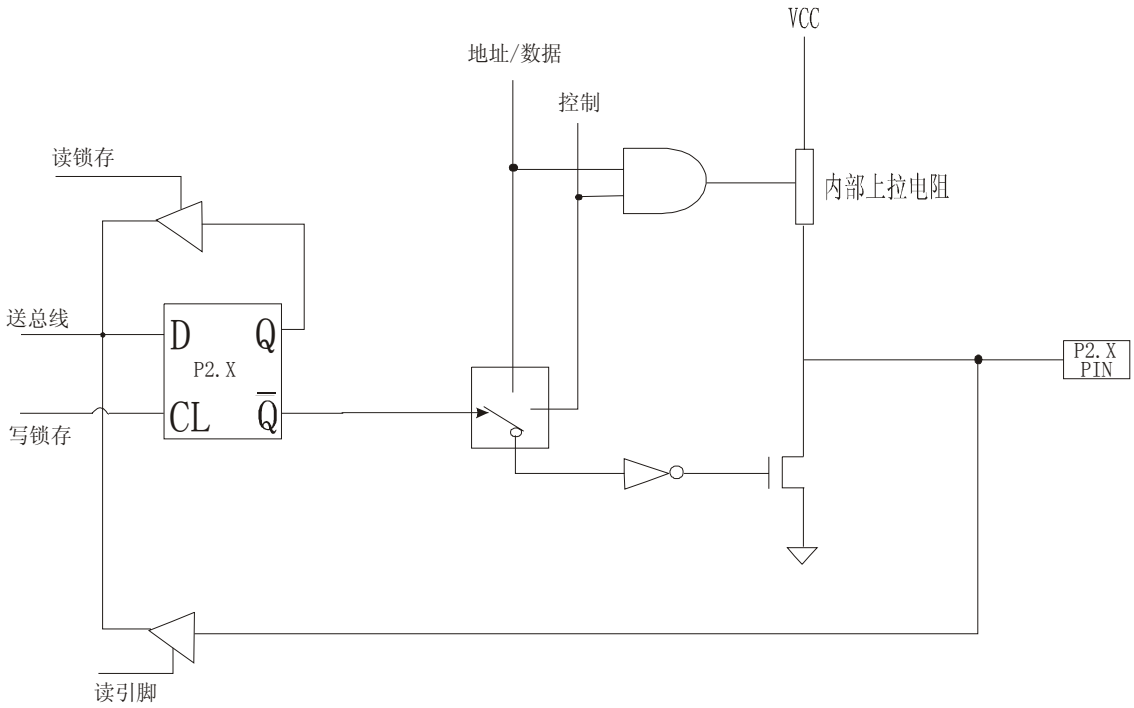
图 2-15 说明四个 I/O 口的典型位锁存和 I/O 缓冲功能。位锁存功能是由 D 触发器实现的，当 CPU 发一个写入锁存信号，则相应的位被锁入 D 触发器。Q 输出放到内部总线上，如果 CPU 发读信号，则 Q 的输出被读入内部总线。



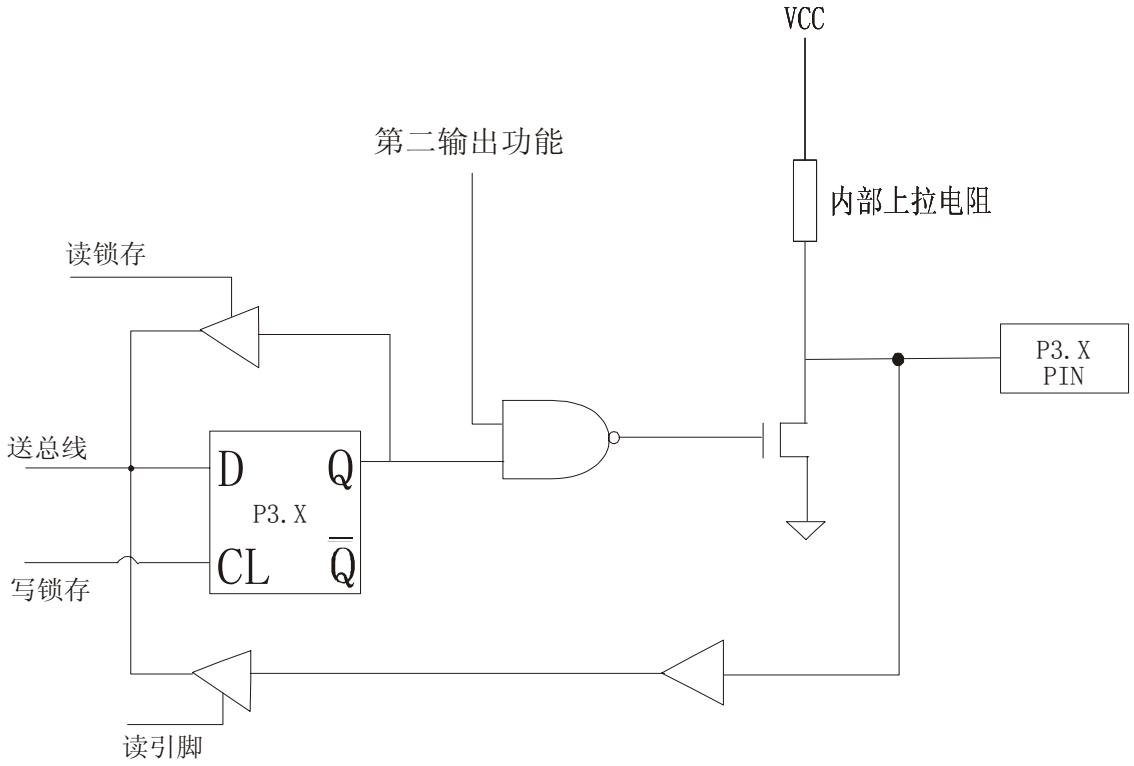
(A)端口0



(B) 端口1



(C)端口2



(D) 端口3

图 2-15 I/O 端口结构

P0 口可以作为通用 I/O 口，同时也作为地址/数据复用总线。P2 口通过控制信号可以在地址总线和通用 I/O 口之间切换，这样可以方便的访问外部 RAM 和 ROM。

P3 口通过与门的控制改变它的输出功能。

P1、P2 和 P3 口都提供了内部上拉电阻，而 P0 口的输出是漏极开路且不提供上拉电阻，因此 P0 口作输入口使用时，必须接上拉电阻且先向 P0 口写 1。否则，在读入端口引脚数据时，由于输出驱动 FET 并接在引脚上，如果 FET 导通就会将输入的高电平拉成低电平，从而产生误读。

2.6 复位及复位操作

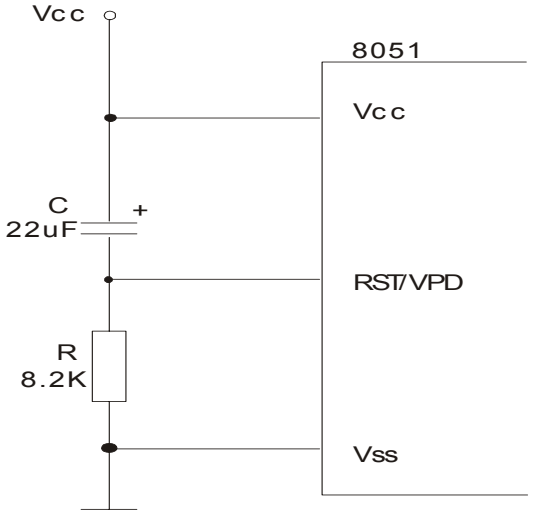
2.6.1 复位操作

RST 引脚有持续 2 个周期的高电平，CPU 复位。复位后 PC 指针指向 0000H，单片机从 0000H 开始执行程序。P0～P3 的复位值为 FFH，SP 的复位值为 07H，而其它 SFR 的复位值均为 0。

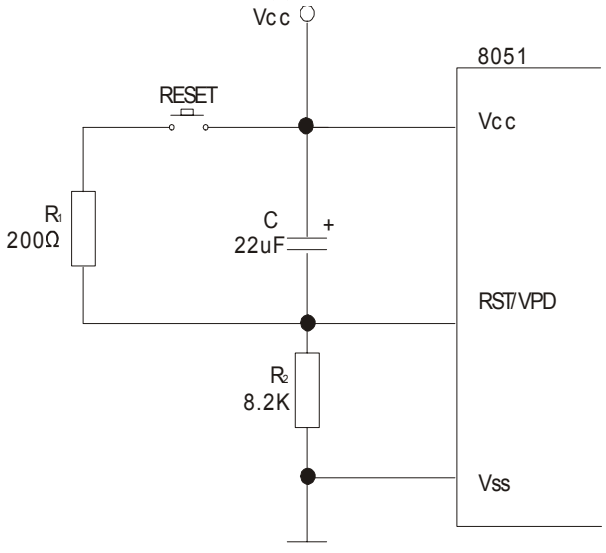
2.6.2 复位电路

复位操作有上电自动复位和按键手动复位两种。

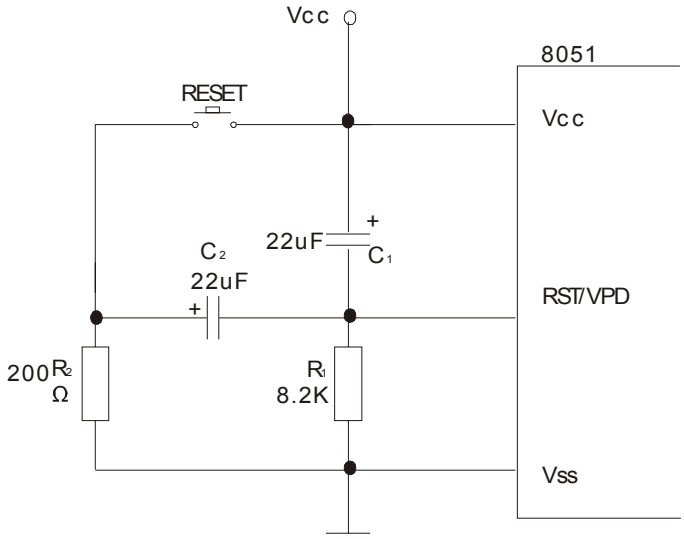
上电自动复位是通过外部复位电路的电容充电来实现的，其电路如图 2-16(a)所示。



+(a) 上电复位



(b) 按键电平复位



（c） 按键脉冲复位

图 2-16 复位电路

这样，只要电源 Vcc 的上升时间不超过 1ms，就可以实现自动上电复位，即接通电源就完成了系统的复位初始化。

按键手动复位有电平方式和脉冲方式。其中，按键电平复位是通过使复位端经电阻与 Vcc 电源接通而实现的，其电路如图 2-16(b)所示；而按键脉冲复位则是利用 RC 微分电路产生的正脉冲来实现的，其电路如图 2-16(c)所示。

复位电路虽然简单，但其作用很重要。一个单片机系统能否正常运行，首先要检查是否能复位成功。目前已有公司推出专用的电路复位芯片，较具代表性的有 Xicor 公司的 X25043/45。

*** 注：**

SST89 系列的单片机提供了软件复位功能, 软件复位是通过将 SFCF[1]（SWR）由‘0’改为‘1’来执行的。软件复位将程序计数器（PC）复位到 0000H。所有的 SFR 寄存器都置为它们的复位值，除了 SFCF[1]（SWR）、WDTC[2]（WDTS）， RAM 的数据将不被改变。

第 三 章 8051 单片机指令系统

尽管目前用 C 语言编程已十分流行，但在有些场合仍离不开汇编语言，譬如对有些 IC 卡的读写时序十分严格，C 语言很难达到要求，这时就必须用汇编语言，并且没有汇编语言基础是很难写出优秀的 C51 程序的。因此，我们还是有必要系统的学习汇编语言的。

3.1 指令的寻址方式

51 单片机的指令系统与 PC 机的指令系统相比，一个明显的特点就是 51 单片机的指令有位寻址功能。位操作部件可以和这些指令结合起来，构成一个完整的位处理器（即布尔处理机），从而大大提高了 51 单片机的位处理能力。指令系统中设计的这个处理布尔变量的指令集，在设计需大量处理位变量的程序时十分有效、方便，可将大量的硬件组合逻辑用软件来代替。

3.1.1 指令系统的寻址方式

指令的一个重要组成部分是操作数，指令给出参与运算的数据的方式称为寻址方式，换句话说，寻址方式就是寻找确定参与操作的数的真正地址。在 51 系列单片机的指令系统中寻址方式共有 7 种，可概括如下：

1. 立即寻址

立即寻址也称为立即数，它是在指令操作数域直接给出参加运算的操作数，其指令格式如下：

例如指令： MOV A，＃70H

这条指令的功能是将立即数 70H 传送到累加器 A 中。

2. 直接寻址

在直接寻址方式中，指令操作数域给出的是参加运算的操作数地址。在 51 单片机中，直接地址只能用来表示特殊功能寄存器、内部数据寄存器和位地址空间。其中特殊功能寄存器和位地址空间只能用直接寻址方式。

例如指令： MOV A，70H

这条指令的功能是将 70H 单元中的数传送到累加器 A 中。

例如指令：ANL 70H，＃48H

表示 70H 单元中的数与立即数 48H 相“与”，其结果存放在 70H 单元中。其中 70H 为直接地址，表示内部数据存储器 RAM 中的一个单元。

3. 寄存器寻址

寄存器寻址是对选定的工作寄存器 R7~R0、累加器 A、通用寄存器 B、地址寄存器和进位 C 中的数进行操作。其中寄存器 R7～R0 由指令码的低 3 位表示，Acc、B、DPTR 及进位位隐含在指令码中。因此在 51 单片机的指令系统中，寄存器寻址也包含一种隐含寻址方式。

寄存器工作区的选择由状态寄存器 PSW 中的 RS1、RS0 来决定。指令操作数域指定的寄存器均指当前工作区中的寄存器。

例如指令：INC R0； （R0）＋1 送 R0

4. 寄存器间接寻址

在寄存器间接寻址方式中，指令操作数给出的是存放操作数地址的寄存器。在 51 单片机的指令系统中，可作为寄存器间接寻址的寄存器有工作寄存器 R0～R1、堆栈指示器 SP 和地址寄存器 DPTR。在指令助记符中，间接寻址用符号@来表示。

例如指令：ANL A，@R0

表示寄存器 R0 中的数所指定的存储单元中的数和累加器 A 中的数相“与”，其结果存放在累加器 A 中。其执行示意图如图 3-1 所示，寄存器工作区由 RS1、RS0 决定。

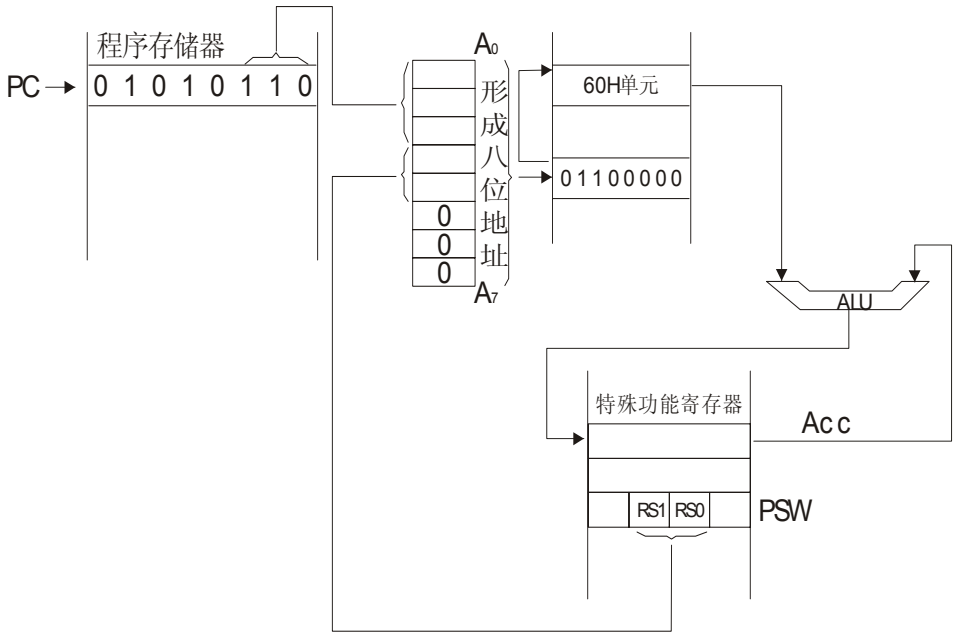


图 3-1 ANL A , @R0 指令执行示意图

5. 相对寻址

相对寻址是将程序计数器 PC 中的当前值与指令第二字节给出的数相加，其结果作为转移指令的转移地址。转移地址也称为转移目的地址，PC 中的当前值称为基地址，指令第二字节给出的数称为偏移量。由于目的地址是相对于 PC 中的基地址而言，所以这种寻址方式称为相对寻址。偏移量为带符号的数，所能表示的范围为＋127~－128.这种寻址方式主要用于转移指令。

例如指令：JC 80H； C＝1 跳转

表示若进位位 C 为 0，则程序计数器 PC 中的内容不改变，即不转移。若进位位 C 为 1，则以 PC 中的当前值为基地址，加上偏移量 80H 后所得到的结果作为该转移指令的目的地址。

6. 变址寻址

在变址寻址方式中，指令操作数域指定一个存放变址基值的变址寄存器。变址寻址时偏

移量与变址基值相加，其结果作为操作数的地址。在 51 单片机的指令系统中，变址寄存器有程序计数器 PC 和地址寄存器 DPTR。

例如指令：MOVC A，@A＋DPTR 表示累加器 A 为偏移量寄存器，其内容与地址寄存器 DPTR 中的内容相加，其结果作为操作数的地址，取出该单元中的数送入累加器 A 中。

7. 位寻址

位寻址是指对一些内部数据存储器 RAM 和特殊功能寄存器进行位操作时的寻址。在进行位操作时，借助于进位位 C 作为位操作累加器，指令操作数域直接给出该位的地址，然后根据操作码的性质对该位进行操作。位地址与字节直接寻址中的字节地址形式完全一样，主要由操作码加以区分，使用时须予以注意。

下表是对以上七种寻址方式的总结。

表 3-1 51 单片机寻址方式

序 号	方 式	使用的变量	寻址空间
1	位寻址		内部RAM和特殊功能寄存器的位地址空间
2	相对寻址	PC＋偏移量	程序存储器
3	变址寻址	A＋DPTR、A＋PC	程序、数据存储器
4	寄存器间接寻址	@R0/@R1/SP	内部 RAM
		@R0/@R1/DPTR	外部 RAM
5	立即寻址		程序存储器
6	直接寻址		内部 RAM 和特殊功能寄存器
7	寄存器寻址	R7～R0/A/B/C/DPTR	

3.1.2 机器指令

我们编程所用的汇编语言虽然很底层，但是机器还是无法读懂的。机器所认识的只是一位一位的二进制代码，譬如 0001101101000。每条汇编指令都对应着一条机器指令，我们看下面的例子。

```
地址        机器码          源程序
                            ORG     0100H
0100H       F0              MOVX    @DPTR , A
            7480            MOV     A , #80H
```

3.1.3 指令分类总结

51 指令系统按功能可以分为：

1. 数据传送指令

2. 算术操作指令
3. 逻辑操作指令
4. 控制转移指令
5. 布尔变量操作指令

按功能分类的指令系统表如表 3-2 所示。

表 3-2 算术运算指令

十六进制代码	助 记 符	功 能	对标志位影响				字节数	周期数
			P	OV	AC	CY		
28~2F	ADD A,Rn	A+Rn→A	√	√	√	√	1	1
25	ADD A,direct	A+(direct)→A	√	√	√	√	2	1
26,27	ADD A,@Ri	A+(Ri)→A	√	√	√	√	1	1
24	ADD A,#data	A+data→A	√	√	√	√	2	1
38~3F	ADDC A,Rn	A+Rn+CY→A	√	√	√	√	1	1
35	ADDC A,direct	A+(direct)+CY→A	√	√	√	√	2	1
36,37	ADDC A,@Ri	A+(Ri)+CY→A	√	√	√	√	1	1
34	ADDC A,#data	A+data+CY→A	√	√	√	√	2	1
98~9F	SUBB A,Rn	A-Rn-CY→A	√	√	√	√	1	1
95	SUBB A,direct	A-(direct)-CY→A	√	√	√	√	2	1
96,97	SUBB A,@Ri	A-(Ri)-CY→A	√	√	√	√	1	1
94	SUBB A,#data	A- data-CY→A	√	√	√	√	2	1
04	INC A	A+1→A	√	×	×	×	1	1
08~0F	INC Rn	Rn+1→Rn	×	×	×	×	1	1
05	INC direct	direct +1→direct	×	×	×	×	2	1
06,07	INC @Ri	(Ri)+1→Ri	×	×	×	×	1	1
A3	INC DPTR	DPTR +1→DPTR	×	×	×	×	1	2
14	DEC A	A-1→A	√	×	×	×	1	1
18~1F	DEC Rn	Rn-1→Rn	×	×	×	×	1	1
15	DEC direct	Direct-1→direct	×	×	×	×	2	1
16,17	DEC @Ri	(Ri)-1→Ri	×	×	×	×	1	1
A4	MUL AB	A×B→AB	√	√	×	0	1	4
84	DIV AB	A/B→AB	√	√	×	0	1	4
D4	DA A	对 A 进行十进制调整	√	√	√	√	1	1

表 3－3 逻辑运算指令

十六进制代码	助 记 符	功 能	对标志位影响				字节数	周期数
			P	OV	AC	CY		

58~5F	ANL A,Rn	A∧Rn→A	√	×	×	×	1	1
55	ANL A,direct	A∧(direct)→A	√	×	×	×	2	1
56,57	ANL A,@Ri	A∧(Ri)→A	√	×	×	×	1	1
54	ANL A,#data	A∧data→A	√	×	×	×	2	1
52	ANL direct,A	(direct) ∧A→(direct)	×	×	×	×	2	1
53	ANL direct,#data	(direct) ∧data→(direct)	×	×	×	×	3	2
48~4F	ORL A,Rn	A∨Rn→A	√	×	×	×	1	1
45	ORL A,direct	A∨(direct)→A	√	×	×	×	2	1
46,47	ORL A,@Ri	A∨(Ri)→A	√	×	×	×	1	1
44	ORL A,#data	A∨data→A	√	×	×	×	2	1
42	ORL direct,A	(direct)∨A→(direct)	×	×	×	×	2	1
43	ORL direct,#data	(direct)∨data→(direct)	×	×	×	×	3	2
68~6F	XRL A,Rn	A⊕Rn→A	√	×	×	×	1	1
65	XRL A,direct	A⊕(direct)→A	√	×	×	×	2	1
66,67	XRL A,@Ri	A⊕(Ri)→A	√	×	×	×	1	1
64	XRL A,#data	A⊕data→A	√	×	×	×	2	1
62	XRL direct,A	(direct) ⊕A→(direct)	×	×	×	×	2	1
63	XRL direct,#data	(direct) ⊕data→(direct)	×	×	×	×	3	1
E4	CLR A	0→A	√	×	×	×	1	2
F4	CPL A	A 取反→A	×	×	×	×	1	1
23	RL A	A 循环左移一位	×	×	×	×	1	1
33	RLC A	A 带进位循环左移一位	√	×	×	√	1	1
03	RR A	A 循环右移一位	×	×	×	×	1	1
13	RRC A	A 带进位循环右移一位	√	×	×	√	1	1
C4	SWAP A	A 半字节交换	×	×	×	×	1	1

表 3-4 数据传送指令

十六进制代码	助 记 符	功 能	对标志位影响				字节数	周期数
			P	OV	AC	CY		
E8~EF	MOV A,Rn	Rn→A	√	×	×	×	1	1
E5	MOV A,direct	(direct)→A	√	×	×	×	2	1
E6,E7	MOV A,@Ri	(Ri)→A	√	×	×	×	1	1
74	MOV A,#data	data→A	√	×	×	×	2	1

F8~FF	MOV Rn, A	A→Rn	×	×	×	×	1	1
A8~AF	MOV Rn,direct	(direct)→Rn	×	×	×	×	2	2
78~7F	MOV Rn,#data	data→Rn	×	×	×	×	2	1
F5	MOV direct,A	A→(direct)	×	×	×	×	2	1
88~8F	MOV direct,Rn	Rn→(direct)	×	×	×	×	2	2
85	MOV direct1,direct2	(direct2)→(direct1)	×	×	×	×	3	2
86,87	MOV direct, @Ri	(Ri)→(direct)	×	×	×	×	2	2
75	MOV direct, #data	data→(direct)	×	×	×	×	3	2
F6,F7	MOV @Ri,A	A→(Ri)	×	×	×	×	1	1
A6,A7	MOV @Ri, direct	(direct)→(Ri)	×	×	×	×	2	2
76,77	MOV @Ri, #data	data→(Ri)	×	×	×	×	2	1
90	MOV DPTR,#data16	data16→(DPTR)	×	×	×	×	3	2
93	MOVC A,@A+DPTR	(A+DPTR)→A	√	×	×	×	1	2
83	MOVC A,@A+PC	(A+PC)→A	√	×	×	×	1	2
E2,E3	MOVX A,@Ri	(Ri)→A	√	×	×	×	1	2
E0	MOVX A,@DPTR	(DPTR)→A	√	×	×	×	1	2
F2,F3	MOVX @Ri,A	A→(Ri)	×	×	×	×	1	2
F0	MOVX @DPTR,A	A→(DPTR)	×	×	×	×	1	2
C0	PUSH direct	SP+1→SP,	×	×	×	×	2	2
		(direct)→(SP)		×	×	×	2	
D0	POP direct	(SP)→(direct),	×	×	×	×	1	2
		SP-1→SP		×	×	×	2	
C8~CF	XCH A,Rn	A←→Rn	√	×	×	×	1	1
C5	XCH A,direct	A←→ (direct)	√	×	×	×	1	1
C6,C7	XCH A,@Ri	A←→(Ri)	√	×	×	×	1	1
D6,D7	XCHD A,@Ri	A0~A3←→(Ri)0~3	√	×	×	×	2	1
C3	CLR C	0→CY	×	×	×	√	1	1
C2	CLR bit	0→bit	×	×	×	√	2	1
D3	SETB C	1→CY	×	×	×	√	1	1
D2	SETB bit	1→bit	×	×	×	√	2	1
B3	CPL C	CY 取反→CY	×	×	×	√	2	1
B2	CPL bit	Bit 取反→bit	×	×	×	√	2	1
82	ANL C,bit	CY∧bit→CY	×	×	×	√	2	2
B0	ANL C,/bit	CY∧/bit→CY	×	×	×	√	2	2
72	ORL C,bit	CY∨bit→CY	×	×	×	√	2	2
A0	ORL C,/bit	CY∨/bit→CY	×	×	×	√	2	2
A2	MOV C,bit	bit→CY	×	×	×	√	2	1
92	MOV bit,C	CY→bit	×	×	×	×	2	2

表 3-5 控制转移指令

十六进制代码	助 记 符	功 能	对标志位影响				字节数	周期数
			P	OV	AC	CY		
	ACALL addr11	PC+2→PC,SP+1→SP,PCL →(SP),SP+1→SP,PCH→(SP),addr11→PC	×	×	×	×	2	2
12	LCALL addr16	PC+3→PC,SP+1→SP,PCL →(SP),SP+1→SP,PCH→(SP),addr16→PC	×	×	×	×	3	2
22	RET	(SP)→PCH,SP-1→SP,(SP) →PCL, SP-1→SP	×	×	×	×	1	2
32	RETI	(SP)→PCH,SP-1→SP,(SP) →PCL, SP-1→SP	×	×	×	×	1	2
	AJMP addr11	PC+2→PC, addr11→PC	×	×	×	×	2	2
02	LJMP addr16	addr16→PC	×	×	×	×	3	2
80	SJMP rel	PC+2→PC, PC+rel→PC	×	×	×	×	2	2
73	JMP @A+DPTR	(A+DPTR) →PC	×	×	×	×	1	2
60	JZ rel	PC+2→PC 若 A=0 PC+rel→PC	×	×	×	×	2	2
70	JNZ rel	PC+2→PC 若 A ≠ 0 PC+rel→PC	×	×	×	×	2	2
40	JC rel	PC+2→PC 若 CY=0 PC+rel→PC	×	×	×	×	2	2
50	JNC rel	PC+2→PC 若 CY ≠ 0 PC+rel→PC	×	×	×	×	2	2
20	JB bit,rel	PC+3→PC 若 bit=1 PC+rel→PC	×	×	×	×	3	2
30	JNB bit,rel	PC+3→PC 若 bit=0 PC+rel→PC	×	×	×	×	3	2
10	JBC bit,rel	PC+3→PC 若 bit=1 ， 0→bit， PC+rel→PC	×	×	×	×	3	2
B5	CJNE A,direct,rel	PC+3→PC 若 A ≠ (direct)， 则 PC+rel→PC；若 A<(direct),则 1→CY	×	×	×	√	3	2
B4	CJNE A,#data,rel	PC+3→PC 若 A ≠ data， 则 PC+rel→PC；若 A< data,则 1→CY	×	×	×	√	3	2
B8~BF	CJNE Rn,#data,rel	PC+3→PC 若 Rn ≠ data， 则 PC+rel→PC;若 Rn < data,则 1→CY	×	×	×	√	3	2
B6~B7	CJNE @Ri,#data,rel	PC+3→PC 若 Ri ≠ data， 则	×	×	×	√	3	2

D8~DF	DJNZ Rn,rel	PC+rel→PC；若 Ri < data,则 1→CY Rn－1→Rn，PC+2→PC, 若 Ri≠0，则 PC+rel→PC	×	×	×	×	2	2
D5	DJNZ direct,rel	(direct)－1→Rn，PC+2→PC, 若 Ri≠0，则 PC+rel→PC	×	×	×	×	3	2
00	NOP	空操作	×	×	×	×	1	1

3.2 伪指令操作

像 PC 机的汇编语言一样，单片机也为汇编的需要提供了汇编伪指令。汇编伪指令是一种控制汇编进程的特殊控制符号，其功能是改变汇编器的状态并将一些必要的信息（如段定义等）加入到目标文件中去。A51 允许采用汇编伪指令进行符号定义，保留和初始化存储器空间，控制程序连接，控制汇编状态和段选择。下面将分别介绍这些伪指令及其功能。

我们发现市面上许多书籍对指令讲的较为详细，然而涉及伪指令却一略而过，针对这种现象，在本书对伪指令给出了很详细的分析。

3.2.1 控制汇编状态的指令

A51 中控制汇编状态的指令可用来控制汇编的结束，设定程序的起始地址以及通知汇编器使用 8051 的哪一个工作寄存器组。控制汇编状态的指令通常有 3 条。

1. ORG 指令

程序定位指令，设定一个程序的起始地址。

指令格式： ORG 地址

用 ORG 指令可以改变地址计数器的值，但不会产生一个新的段，因此当前段中可能会存在地址空隙。

例如：

```
        ORG     0000H
        JMP     MAIN
MAIN:
        ORG     0100H
```

在当前段中存在地址空隙。

2. USING 指令

USING 指令通知汇编器使用 8051 的哪一个工作寄存器。

指令格式：USING 数字（0~3）

例如：USING 2

表示设置第 2 组工作寄存器为当前工作寄存器，一般默认值为 0。

3. END 指令

END 指令表示汇编结束，有的编译器会提醒程序员在程序结束时加上 END，而有的编译器会自动给程序添上 END。

END 在程序中只能出现一次，它后面的行将被忽略。

3.2.2 符号定义指令

符号定义指令共 8 条，用来声明再定位及其类型，定义各种符号名并赋以存储器地址。

1. EQU 指令

EQU 指令用于将一个数值或寄存器名赋给一个指定符号名。

指令格式： 符号名 EQU 表达式
符号名 EQU 寄存器名

经过 EQU 指令赋值的符号可在程序的其它地方使用，以代替其赋值。

例如：MAX EQU 2000

则在程序的其它地方出现 MAX，就用 2000 代替。

2. SET 指令

SET 指令类似于 EQU 指令，不同的是 SET 指令定义过的符号可重定义。

指令格式：符号名 SET 表达式
符号名 SET 寄存器名

例如：MAX SET 2000
MAX SET 3000

3. BIT 指令

BIT 指令用于将一个位地址赋给指定的符号名。

指令格式：符号名 BIT 位地址

经 BIT 指令定义过的位符号名不能更改。

例如：X_ON BIT 60H ；定义一个绝对位地址
X_OFF BIT 24h.2 ；定义一个绝对位地址

4. DATA 指令

DATA 指令用于将一个内部 RAM 的地址赋给指定的符号名

指令格式：符号名 DATA 表达式

数值表达式的值应在 0~255 之间，表达式必须是一个简单再定位表达式。

例如：REGBUF DATA 40H
PORT0 DATA 80H

5. XDATA 指令

XDATA 指令用于将一个外部 RAM 的地址赋给指定的符号名。

指令格式：符号名 XDATA 表达式

```
例如：RSEG    XSEG1                ；选择一个外部数据段
      ORG     100H
      MING    DS      10           ；在标号 MING 处保留 10 个字节
      HOUR    XDATA   MING+5
      MUNIT   XDATA   HOUR+5
```

6. IDATA 指令

IDATA 指令用于将一个间接寻址的内部 RAM 地址赋给指定的符号名。

指令格式：符号名 IDATA 表达式

例如：FULLER IDATA 60H

7. CODE 指令

用于将程序存储器 ROM 地址赋给指定的符号名。

指令格式：符号名 CODE 表达式

例如：RESET CODE 00H

8. SEGMENT 指令

SEGMENT 指令用来声明一个再定位段和一个可选的再定位类型。

指令格式：再定位段型 SEGMENT 段类型（再定位类型）

段类型用于指定所声明的段将处的储存器地址空间，可用的段类型有 CODE /XDATA/DATA/IDATA 和 BIT。

```
例如： FLAG     SEGMENT   BIT
       PONITER  SEGMENT   IDATA
```

3.2.3 保留和初始化存储器空间

此指令用于在存储器空间内保留和初始化字、字节和位单元，保留空间始于当前地址的绝对段和当前偏移地再定位段。

1. DS

以字节为单位在内部和外部存储器保留存储器空间。

指令格式：[标号：] **DS** 数值表达式

DS 指令使当前数据段的地址计数器增加表达式结果之值，地址计数器与表达式结果之和不能超过当前地址空间。标号值将是保留区的第一个字节地址。

例如：ORG 0200H

CUNTER DS 10

COUNTER 的地址是 0200H。

2. DBIT

在内部数据区的 BIT 段以位为单位保留存储空间。

指令格式：[标号：] **DBIT** 数值表达式

其操作类似于 DS。

3. DB

以给定表达式的值的字节形式初始化代码空间。

指令格式：[标号：] **DB** 数值表达式

其操作类似于 DS。

4. DW

以给定表达式的值的双字节形式初始化代码空间。

指令格式：[标号：] **DB** 数值表达式

其操作类似于 DS。

3.2.4 控制连接指令

控制连接伪指令共 3 条，用于表明当前模块中需要使用的外部函数名及可被其它模块调用的函数名，当该函数用于让 C 调用时，声明时前要加下划线“_”。

1. PUBLIC

声明可被其它模块使用的公共函数名。

指令格式：PUBLIC 符号[，符号，符号[，••••••]]

PUBLIC 后可跟多个函数名，用逗号格开。每个函数名都必须是在模块内定义过的。

例如：PUBLIC INTER，_OUTER

其中_OUTER 可供 C 调用。

2. EXTRN

EXTRN 是与 PUBLIC 配套使用的，要调用其它模块的函数，就必须先在模块前声明

指令格式：EXTRN 段类型（符号，符号 ••••••）

例如：EXTRN CODE(TONGXING,ZHUANHUAN)

调用外部 TONGXING 和 ZHUANHUAN 程序。

3. NAME

用来给当前模块命名。

指令格式：NAME 模块名

例如：NAME TIMER

定义一个模块名为 TIMER 的模块。

3.2.5 段选择指令

用来选择当前段是绝对段还是再定位段，使用段选择指令。

1. 绝对段选择指令

绝对选择指令有 CSEG/DSEG/XSEG/ISEG 和 BSEG，分别选择绝对代码段、内部绝对数据段、外部绝对数据段、内部间接寻址绝对数据段和绝对位寻址数据段。

指令格式如下：

CSEG [AT 绝对地址表达式]

DSEG [AT 绝对地址表达式]

XSEG [AT 绝对地址表达式]

ISEG [AT 绝对地址表达式]

BSEG [AT 绝对地址表达式]

2. 再定位段选择指令

再定位段选择指令为 RSEG，用于选择一个已在前面定义过的再定义段作为当前段，

指令格式：RSEG 段名

段名必须是在前面声明过的再定位段。

例如：

```
DATAS       SEGMENT   DATA      ：声明一个再定位 DATA 段
CODES       SEGMENT   CODE      ；声明一个再定位 CODE 段
            BSEG      AT   60H
            RSEG      CODES     ；选择前面声明的再定位 CODE 段作为当前段
```

第 四 章 单片机汇编程序设计

计算机在完成一项工作时，必须按顺序执行各种操作。这些操作是程序设计人员用计算机用计算机所能接受的语言把解决问题的步骤事先描述好的，也就是事先编制好计算机程序，再由计算机去执行。通常，在科学计算中采用高级语言，在实时控制中采用汇编语言。单片机常用于自动控制，而自动控制是实时控制，因此，我们以 51 单片机系列为例，介绍其汇编语言程序设计。本章先介绍编程步骤方法与技巧，然后再给出汇编语言程序设计实例。

4.1 编程方法及步骤

本节讲述如何进行程序设计。也就是面对一个现实的问题我们应该怎样解决，我们怎样把一个实际的任务变成程序模块，然后组合模块，最终实现编程。好的编程方法会使你的工作事半功倍。

编程一般可分为五步。

一、分析问题

首先，要对需要解决的问题进行分析，以求对问题有正确的理解。例如，解决问题的目的是什么，最终要达到什么要求，现有的条件，已知的数据，对运算精度和速度方面的要求等等。

弄清问题是整个工作的基础，在解决科研和生产中的实际问题时，这是不可缺少的一步。但在处理比较简单的问题时，解题的要求往往是显而易见的，无需多加分析。

二、确定算法和输入输出方式

解决一个问题，常常有几种可以选择的方法。从数学的角度来描述，可能有几种不同的算法。在编制程序以前，先要对不同的算法进行分析比较，找出最适宜的算法。然而，哪种算法最佳不是绝对的，比如，用迭代法解微分方程，则要考虑收敛的快慢，在一定的时间里，能否达到所要求的精度。有的问题受内存容量的限制，而对时间要求并不苛刻，在这种情况下，速度较慢而节省内存的算法可能被优先选择。

对于输入方式，在一般情况下是由数据种类、性质及系统中提供的输入设备类型决定的。对运算结果的输出要根据结果的利用方式和系统中提供的设备类型来决定的，如用打字机、X—Y 记录仪或终端等等。

有时，输入输出的方式是由问题本身决定的。比如，如果计算机控制的对象是送来的模拟信号，则要通过模数转换接口电路转换成数字信号送入计算机，经过处理后，再通过数模转换接口电路变成模拟信号送给控制对象。这样，输入输出主要是通过模数、数模转换实现的。

三、程序结构设计

程序结构设计是把研究课题转换为程序的准备阶段。如果程序较小且简单，此阶段可能仅仅是绘制一张流程图。如果程序较大或较复杂，设计者就要考虑较为完善的方法。这时，我们通常考虑使用模块化的程序设计方法。

实际的应用程序一般都由一个主程序（包括若干个功能模块）和多个子程序构成。每一程序模块都能完成一个明确的任务，实现某个具体功能，如发送、接收、延时、打印等。采用模块化的程序设计方法，由下述优点：

1、 单个模块结构的程序功能单一，易于编写、调试和修改。

2、 便于分工，从而可使多个程序员进行程序的编写和调试工作，加快软件研制进度。

3、 程序可读性好，便于功能扩充和版本升级

4、 对程序的修改可局部进行，其它部分可以保持不变。

5、 对于使用频繁的子程序可以建立子程序库，便于多个程序调用

在进行模块划分时，应首先弄清每个模块的功能，确定其数据结构以及与其它模块的关系；其次是对主要任务进行细化，把一些专用的子任务交由下一级即第二级子模块完成，这时也需要弄清它们之间的关系。按这种方法一直细分成易于理解和实现的小模块为止。

模块的划分有很大的灵活性，但也不能随意划分。划分模块时应遵循下述原则：

2、 每个模拟应具有独立的功能，能产生一个明确的结果，这就是单模块的功能高内聚性。

3、 模块之间的控制耦合应尽量简单，数据耦合应尽量少，这就是模块间的低耦合性。

控制耦合是指模块进入和退出 的条件和方式，数据耦合是指模块间的信息交换（传递）方式、交换量的多少及交换频繁的程度。

4、 模块长度适中。模块语句的长度通常在 20 条～100 条的范围较为合适。模块太长时，分析和调试比较困难，失去模块化程序结构的优越性；过短，则模块的连接太复杂，信息交换太频繁，因而也不合适。

四、撰写汇编源程序

汇编源程序的编辑可以放在任何文字编辑器里进行，一般的厂家提供的仿真系统都有一个编辑器但都不大好用，现在流行一种程序撰写软件 UltraEdit-32，是专为写程序而设计的。用来进行写 C 语言和汇编源程序都挺方便的，这个软件大家有兴趣的话，可以 去网上下载。

五、错误修改及调试

如果你用的是 C 语言，你可以去用 uVision51 去调试，但建议使用你的供应商提供的调试系统，因为可能你用 uVision51 调试的程序在目标板上就跑不起来，而供应商的软件包有较好的兼容性。一般情况下，汇编语言的兼容性就更需要注意了，有时同一公司的不同版本都不互相兼容。

我们公司提供的开发系统可以兼容 keil C51，大家完全可以在 keil C 的环境中去调试和编译。

4.2 程序设计举例

这一节，我们将讲述 51 单片机常见的程序设计结构并给出应用实例，这些结构有定时程序、查表程序、数据极值查找程序、排序程序、检索程序等。

4.2.1 汇编程序的基本结构

程序设计共有 4 个基本结构，顺序、分支、循环和子程序结构。

4.2.1.1 顺序结构

顺序是最简单的程序结构，在这种结构下，程序按顺序一条一条的执行指令，即不跳转，也不调用子程序。

例：多字节数相加。

由于 51 指令系统中只有单字节加法指令，因此对多字节加法运算，必须从低位字节开始分字节进行。除最低字节可以使用 ADD 指令之外，其它字节相加时，要把低字节的进位考虑进去，故应使用 ADDC 指令。

假定三字节无符号数相加，其中一个加数在内部 RAM 的 50H、51H、和 52H 单元中，另一个加数在内部 RAM 的 53H、54H 和 55H 单元中，要求把相加之和存放在 50H、51H 和 52H

单元中，进位存放在位地址区的 00H 位中。

```
MOV     R0,52H          ；一个加数的低字节地址
MOV     R1,55H          ；另一个加数的低字节地址
MOV     A,@R0
ADD     A,@R1           ；低字节相加
MOV     @R0,A           ；存低字节相加结果
DEC     R0
DEC     R1
MOV     A,@R0
ADDC    A,@R1           ；中间字节带进位相加
MOV     @R0,A           ；存中间字节相加结果
DEC     R0
DEC     R1
MOV     A,@R0
ADDC    A,@R1           ；高字节带进位相加
MOV     @R0,A           ；存高字节相加结果
MOV     00H,C           ；进位送 00H 位保存
```

4.2.1.2 分支结构

在 51 指令系统中，通过条件判断实现分支程序的转移，这些条件判断指令有 JZ/JNZ/CJNE/DJNZ 等，此外，51 指令系统还提供了位转移指令，这些指令有 JC/JNC/JB/JNB/JBC 等。使用这些指令，可以完成或为 0、1，或为正负，以及相等不相等各种条件判断。

分支程序结构图可以如下图描述：

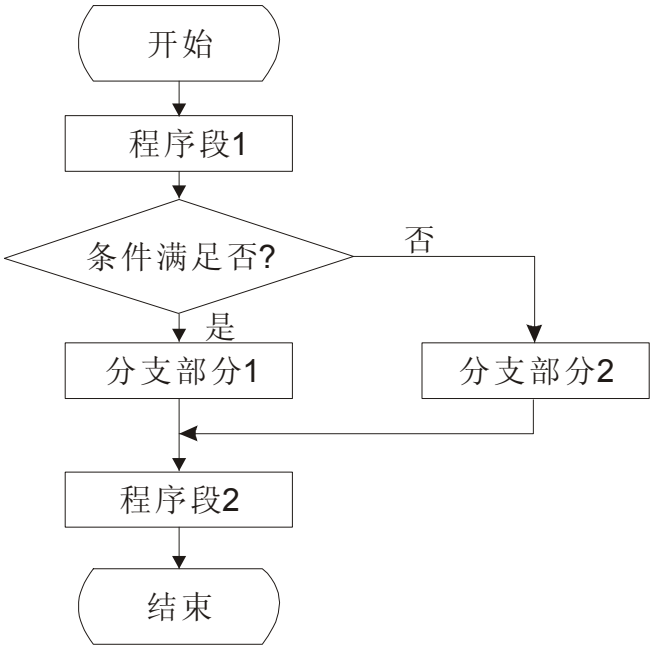


图 4－1 简单分支结构流程图

4.2.1.3 循环结构

循环是为了重复执行某个程序段，通常把这个程序段称为循环体。循环是在一定的条件控制下实现的，为此需要进行条件判断，以决定是继续循环还是退出循环。汇编语言中没有专门的循环指令，在汇编语言程序中，程序循环都是通过条件转移指令实现的。

循环结构可由下图描述：

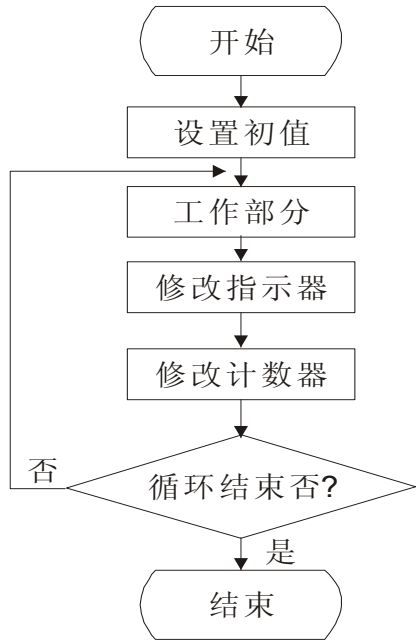


图 4—2 典型的循环程序流程图

4.2.1.4 子程序结构

汇编语言为子程序调用提供了两条指令 ACALL 和 LCALL。在设计子程序时，应特别注意保持堆栈的压入和弹出的均衡，否则不能正确的返回，一般子程序用 RET 返回，而中断子程序要用 RETI 返回。

子程序结构可用下图描述：

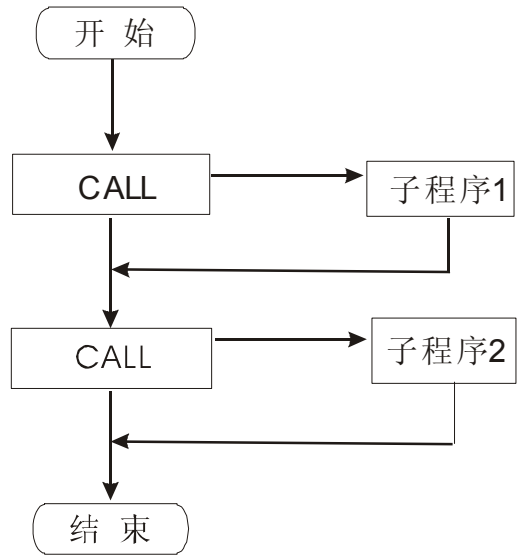


图 4-3 子程序结构图

在单片机的控制应用中，常需要定时，如定时中断、定时检测和定时扫描等。定时功能除可用定时器/计数器实现外，更多的是使用定时程序完成。

定时程序是典型的循环程序，它是通过一个具有固定延时时间的循环程序来实现定时的，因此也把定时程序称为延时程序。定时程序只能使用汇编语言编写。

1．单循环定时程序

下面就是一个最简单的单循环定时程序。

```
        MOV     R5,#TIME
LOOP:   NOP
        NOP
        DJNZ    R5,LOOP
```

NOP 指令的机器周期为 1，DJNZ 的机器周期为 2，故一次循环共 4 个机器周期。若单片机的晶振频率为 6MHz，即一个机器周期是 2us。则一次循环的延时时间为 8us。上面程序总的延时为 8×TIME(us)。本程序的实际延时时间取决于装入寄存器 R5 的定时时间常数 TIME。R5 是 8 位寄存器，故这个程序的最长定时时间为 256×8＝2048us，即定时范围是 8～2048us。可见单循环定时程序的时间延迟较小。

2．较长时间的定时程序

为加长定时时间，通常采用多重循环的方法。例如下例双重循环的定时程序，最长可延时 262914 个机器周期，即 525818us 或大约 526ms（6MHz 晶振频率）。

```
        MOV     R4,#TIME1
LOOP2:
        MOV     R5,#TIME2
LOOP1:
        NOP
        NOP
        DJNZR5,LOOP1
        DJNZ    R4,LOOP2
```

最大定时时间计算公式为：

（256×4＋2＋1）×256＋2＋4＝525828us

3．提高定时精度

单片机是按照严格的固定时序执行指令，因此定时程序的延迟时间是执行循环程序段所需时间的整数倍。

在通常的应用场合下，总是根据需要预先确定定时时间，然后再设计定时程序。为了确保定时时间与循环程序段执行时间之间的整数倍关系，往往要对循环程序段通过增减指令的办法对时间进行微调。

例如如下的定时程序：

```
        MOV         R0,#TIME
```

```
LOOP:   ADD     A,R1
        INC     DPTR
        DJNZ    R0,LOOP
```

由于 ADD 指令的机器周期数为 1，INC 的机器周期为 2，DJNZ 的机器周期为 2，故在 6MHz 的晶振频率下，该程序的定时时间为 10×TIME（us）。

假定要求定时时间为 24us，对于这个定时程序，无论 TIME 取何值均得不到定时时间。通过增加一条 NOP 指令，把循环周期增加到 6，即

```
        MOV     R0,#TIME
LOOP:   ADD     A,R1
        INC     DPTR
        NOP
        DJNZ    R0,LOOP
```

这时，只要 TIME 值设定为 2，就可以精确的得到 24us 的定时。

上述程序中我们用到了 NOP 指令，这条指令什么也不作，只起到微调的作用，注意，如果使用其它指令微调时，一定不要破坏有用存储单元的内容。

4．用一个基本的延时程序满足不同的定时要求

如果一个系统有多个定时要求，我们就可以设计一个基本的延时程序，使其延时时间为各定时时间的最大公约数。然后以此基本程序作为子程序，通过调用的方法实现所需要的不同定时。例如要求的定时时间为 5s、10s 和 20s，设计一个 5s 的延时子程序，则不同的调用情况如下：

```
        MOV     R0,#01H         ；5s 延时
LOOP1:
        LCALL   DELAY
        DJNZ    R0,LOOP1
          •
          •
          •
        MOV     R0,#02H         ；10s 延时
LOOP1:  LCALL   DELAY
        DJNZ    R0,LOOP1
          •
          •
          •
        MOV     R0,#04H         ；20s 延时
LOOP1:  LCALL   DELAY
        DJNZ    R0,LOOP1
          •
          •
          •
```

不同的时间延时是通过给定不同的调用次数来实现的。此外，也可以通过给定基本定时程序以不同的时间常数的方法，实现不同的时间定时。

4.2.3 查表程序

在计算机控制中，查表程序有很重要的应用，常用于实现非线性修正、非线性函数转换以及代码转换等。

51 指令系统提供了专门的查表指令：MOVC A,@A+DPTR 和 MOVC A,@A+PC。这两条指令可提高查表速度，缩短程序长度，使用起来也十分方便，但 在具体使用上有所侧重。

MOVC A,@A+DPTR 适用于在 64KROM 范围内查表。编写查表程序时，首先把查表的首地址送入 DPTR，再将要查找的数据序号（或）下标值送入 A，然后就可以使用该指令进行查表操作，并把结果送入累加器 A。

MOVC A,@A+PC 适用于在在本地范围内查表。编写查表程序时，首先把要查找的数据序号（或）下标值送入 A，再把从查表指令到表的首地址间的偏移量与 A 值相加，然后就可以使用该指令进行查表操作，并把结果送入累加器 A。

例：将一位十六进制数转换成 ASCII 码。

解：本程序中，由 R0 指出十六进制数的存放单元，经转换后结果仍存于原处。

```
HEXASC1:  MOV    A , @R0          ；取十六进制数
          ANL    A , #0FH         ；屏蔽高四位
          ADD    A , #03H         ；修正偏移量
          MOVC   A , @A+PC        ；查表，取得 ASCII 代码
          XCH    A , @R0          ；存储
          INC    R0               ；更新地址
          RET
ASCTAB:   DB     30H,31H,32H,33H,34H
          DB     35H,36H,37H,38H,39H
          DB     41H,42H,43H,44H,45H,46H
```

4.2.4 数据极值查找程序

极值查找就是在给定的数据区中挑出最大值或最小值。

例：内部 RAM 20H 单元开始存放 8 个数，找出其中最大的数。

极值查找操作的主要内容是进行数值大小的比较。假定在比较的过程中，以 A 存放大数，与之逐个比较的另一个数放在 2AH 单元。比较结束后，把查找到的最大的数送到 2BH 单元中。

程序流程如图 4—4 所示，程序如下

```
MOV   R0   ，#20H
MOV   R7   ，#08H
```

```
                MOV   A，@R0
                DEC    R7
LOOP：
                INC    R0
                MOV   2AH，@R0
                CJN   A，2AH，CHK
CHK：           JNC    LOOP1
                MOV    A，@R0
LOOP1：
                DJNZ     R7，LOOP
               MOV    2BH，A
HERE：         AJMP    HERE
```

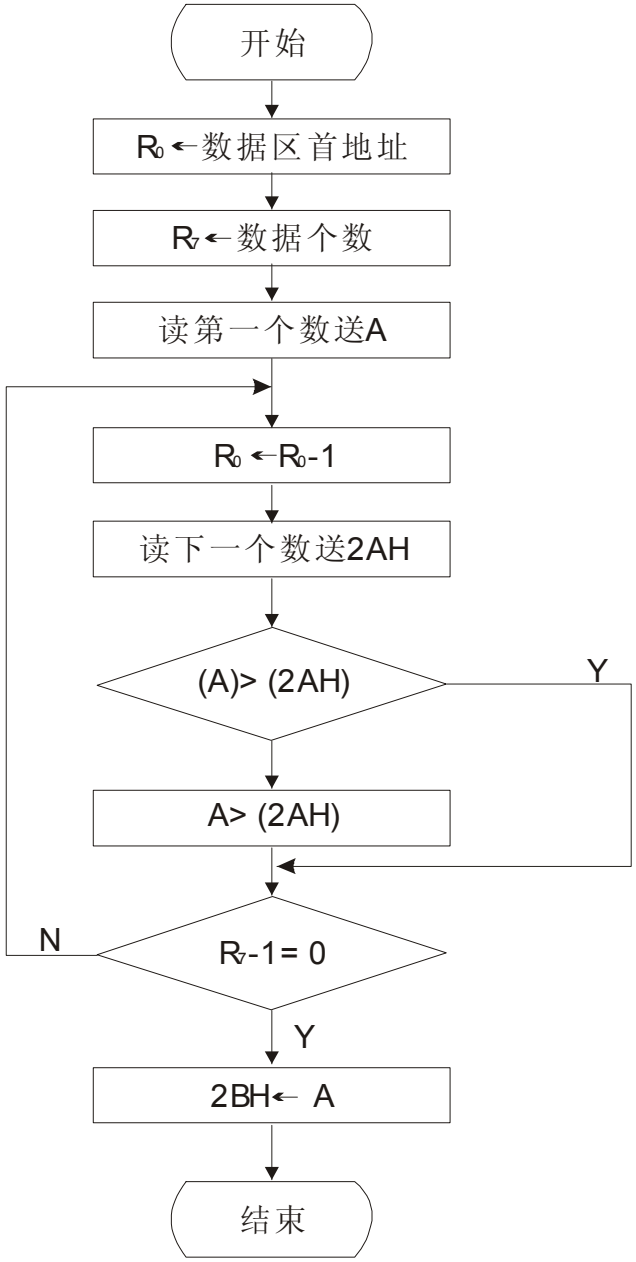


图 4－4　极值查找程序流程

4.2.5 数据排序程序

1. 算法说明

数据排序的算法很多，常用的有插入，冒泡，快速，选择，堆积，二路归并排序法以及其基数排序法。现以冒泡排序法为例，说明数据升序算法及编程实现。

冒泡法是一种相邻数互换的排序方法，因为其过程类似水中汽泡的上浮，故称其为冒泡法。执行时从前向后进行相邻比较，如数据的大小次序与要求的次序不符，就将两个数互换，否则为正序不互换。假定是升序，则通过这种相邻数互换的方法，使小数向前移，大数向后移。如此从前向后的进行一次冒泡，就会把最大的数换到最后。再进行一次冒泡，就会把次大的数排在倒数第二的位置。

例如原始的数据的顺序是：50，38，7，13，59，44，78，22

第一次冒泡的过程是：

50，38，7，13，59，44，78，22，（逆序 互换）
38，50，7，13，59，44，78，22，（逆序 互换）
38，7，50，13，59，44，78，22，（逆序 互换）
38，7，13，50，59，44，78，22，（正序 不互换）
38，7，13，50，59，44，78，22，（逆序 互换）
38，7，13，50，44，59，78，22，（正序 不互换）
38，7，13，50，44，59，78，22，（逆序 互换）
38，7，13，50，44，59，78，22，（第一次冒泡结束）

如此进行各次冒泡的结果是：

第一次：38,7,13,50,44,59,22,78
第二次:7,13,38,44,50,22.59,78
第三次:7,13,38,44,50,22.59,78
第四次:7,13,38,22,44,50,59,78
第五次:7,13,22,38,44,50,59,78
第六次:7,13,22,38,44,50,59,78
第七次:7,13,22,38,44,50,59,78

可以看出，冒泡程序到第五次已经完成。

针对上述冒泡排序的过程，有两个问题需要说明 1

1. 由于每次冒泡都从前向后排定一个大数（假定升序），因此每次冒泡所需进行的比较次数都递减 1，例如有 N 个数排序，则第 1 次冒泡需比较（N-1）次，第 2 次需（N-2）次，依此类推。但实际编程时，为了简化程序，往往把各次的比较次都固定为（N-1）次，尽管有许多重复操作也在所不惜。

2. 对于 N 个数，理论上说应进行（N-1）次冒泡才能完成排序，但实际上常常不到（N-1）次就已排好序，如本例共 8 个数，按说应进行 7 次冒泡，实际上进行到第 5 次时排序就完成

了。判定排序是否完成的最简单方法是看各次冒泡中有无互换发生。如果有数据互换，说明排序还没完成，否则就表示已排好序。为此，控制排序常不使用计数方法。而使用设置互换的标志的方法，以其状态表示在一次冒泡中有无数据互换。

2．程序设计

假定 8 个数据连续存放在 20H 为首地址的内部 RAM 单元中，使用冒泡法进升序排序编程。

设 R7 为比较次数器，初始为 07H。TR0 为冒泡过程中是否有数据互换的状态标志，TR0=0 表明无互换发生，TR0=1 表明有互换发生。

按前述冒泡排序算法，流程如图 4－5 所示。

```
SORT:    MOV    R0   , #20H
         MOV    R7   ,#07H
         CLR    TR0
LOOP:
         MOV    A    , @R0
         MOV    2BH, A
         INC    R0
         MOV    2AH,@R0
         CLR    C
         SUBB   A,   @R0
         JC     NEXT
          MOV   @R0, 2BH
          DEC   R0
          MOV   @R0,2AH
          INC   R0
          SETB  TR0
NEXT:
         DJNZ   R7,    LOOP
          JB    TR0,SORT
HERE      SJMP  $
```

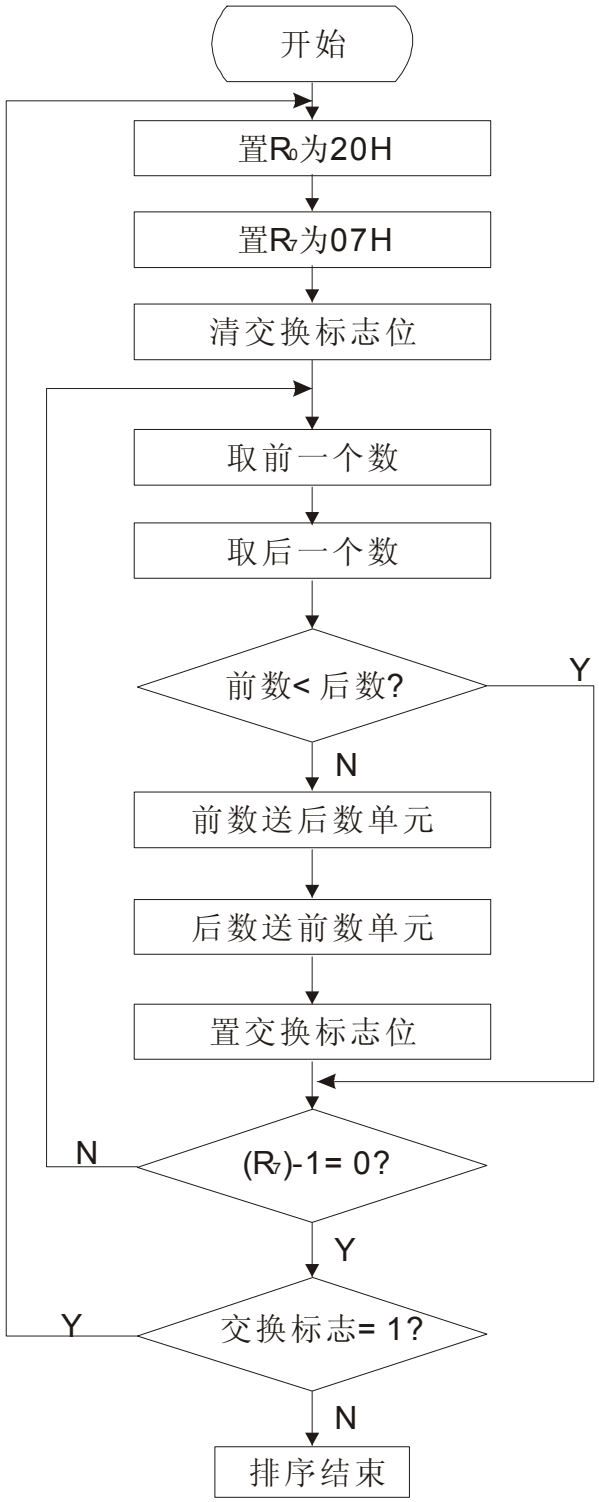


图 4－5 冒泡法排序程序流程

4.2.6 数据检索程序

数据检索程序是在数据区中查找关键字的操作，有两种数据检索方法，即顺序检索和对分检索。下面分别介绍

1． 顺序检索

所谓顺序检索就是把数据区中的数据从前向后逐个比较，判断是否相等。

[例] 假定数据首地址是内部 RAM 20H 单元，数据区长度为 8，关键字为 8，关键字放在 2BH 单元，检索成功的数据的数据序号放在 2CH 单元。

检索开始时应把 2CH 单元初始化为 00H。程序运行结束后，如 2CH 单元的内容仍为 00H，则表示没有检到关键字；否则检索成功，2CH 单元的内容即为关键字在数据区中的序号。（从 0 开始）。程序如下：

```
        MOV   R0,   #20H
        MOV   R7,   #08H
        MOV   2CH, #00H
        MOV   R2 , #00H
        MOV   2BH,#KEY
NEXT:
        INC   R2
        MOV   2AH,@R0
        CLR       C
        MOV   A,2BH
        SUBB   A,@R0
        JZ      ENDP
        INC    R0
        DJNZ   R7,   NEXT
        MOV   R2,   #00H
ENDP:
        MOV    2CH,   R2
HERE:   AJMP    HERE
```

2． 对分检索

对分检索的前提是数据已排好序（假定是升序）。对分检索是按对分的原则取数进行关键字比较，具体的过程是：取数组中间的位置与关键字比较，如果相等则检索成功；如果取数大于关键字，则下次 对分检索的范围是从数据区的起点到本次取数；如果取数小于关键字，则下次对分检索的范围是从本次取数到数据区终点。依此类推，逐次缩小检索范围，直到最后。

对分检索可以减少检索次数，大大提高数据检索速度。但对分检索是一种递归算法。具体实现时首先要确定检索范围。范围的起点是 0，而终点是把最后一个数的序号加 1，这样才能使最后的一个数也处在有效检索范围之内。这是因为，在程序中对分序号是通过起点与终

点相加，然后除 2 取整而得到的。

对分检索程序流程如图 4－6 所示。

[例]假定检索数据区在内部 RAM 中，首地址为 data，其数据为无符号数，并已按升序排序。工作单元定义如下：

2AH 存放检索范围的起

2BH 存放检索关键字

R0 先指向数据区首地址，检索开始后，则为对分读数地址

R2 检索成功的标志，如检索成功，则数据序号放入其中；否则置为 FFH 状态

R3 检索次数计数器

R4 存放检索到的数据

R7 存放检索范围的终点

程序如下:

```
        MOV    2AH,#00H
        MOV    R7,   #DVL
        MOV    2BH, #KEY
        MOV    R3,   #01H
LOOP1:
        MOV   R0,   #DATA
        MOV    A,    2AH
        ADD    A,    R7
        CLR    C
        RRC    A
        MOV    R2,    A
        CLR    C
        SUBB   A,    2AH
        JZ     LOOP3
        MOV    A, R2
        ADD    A, R0
        MOV    R0,   A
        MOV    A, @R0
        MOV    R4, A
        CLR    C
        SUBB   A, 2BH
        JZ     LOOP5
        JNZ    LOOP2
        MOV    2AH,R2
        INC    R3
        SJMP   LOOP1
```

```
LOOP2:
        MOV     A , R2
        MOV     R7,     A
        INC     R3
        SJMP    LOOP1
LOOP3:
        MOV     R0,#DATA
        MOV     A ,@R0
        CJNE    A , 2BH, LOOP4
        MOV     R4,     A
        SJMP    LOOP
LOOP4:
        MOV     A, #FFH
        MOV     R2,     A
LOOP5:
        SJMP    LOOP5
```

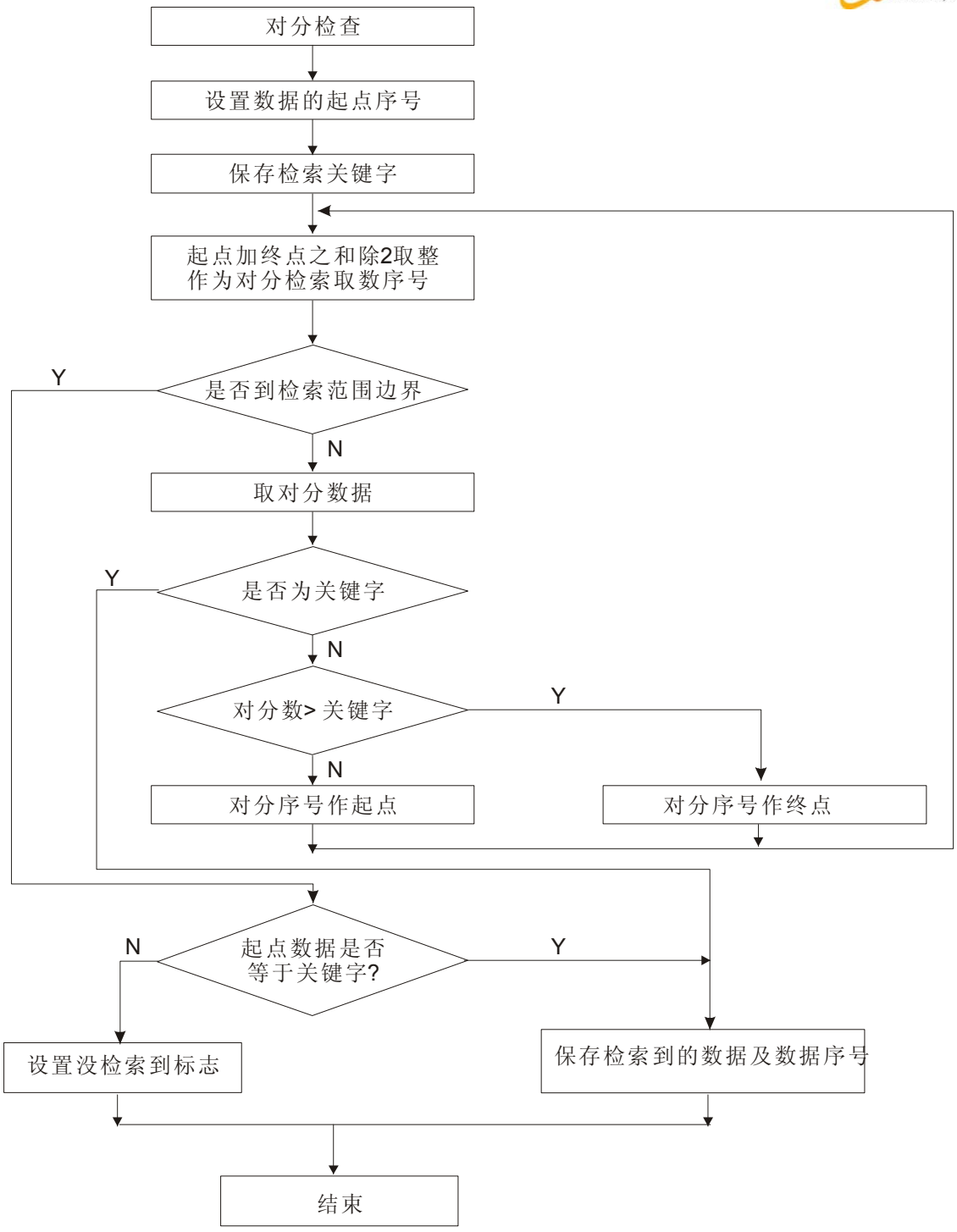


图 4－6　对分检索程序流程

第 五 章 中 断 系 统

中断系统是为使 CPU 对外界突发事件的处理能力而设置的。当 CPU 正在处理某个事件的时候外界发生了紧急事件请求，要求 CPU 暂停当前的工作，转而去处理这个紧急事件。处理完后，再回到原来被中断的地方，继续原来的工作，这样的过程称为中断。实现这种功能的部件称为中断系统，请求中断的请求源称为中断源。中断源有优先级别，优先级别高的先被 CPU 响应。关于中断的更详细的说明请参阅计算机组成原理。

5.1 中断请求源

8031/8051/8751 单片机提供 5 个中断源，其中两个为外部中断源，由 INT0、INT1（P3.2,P3.3）输入；另外两个是片内定时器/计数器溢出中断 T0/C0,T1/C1；还一个是片内串行口中断。

中断系统的结构示意图如下所示：

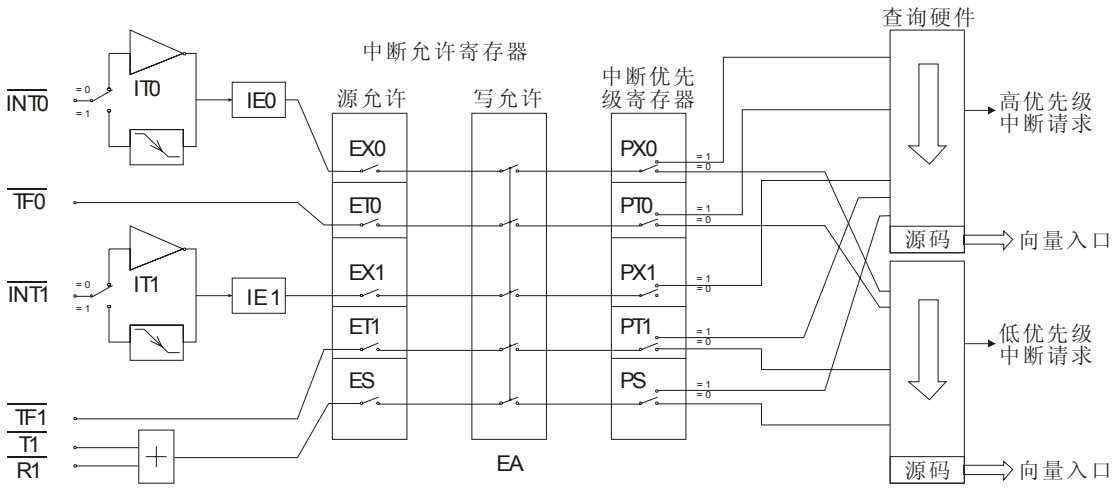


图5－1 MCS－51中断系统结构示意图

相应 5 个中断源的入口地址见下表

中断源	入口地址
外部中断 0	0003H
定时器/计数器 T0	000BH
外部中断 1	0013H
定时器/计数器 T1	001BH
串行口中断	0023H

*** 注：**

SST 单片机最少提供 6 个中断源，最多有 9 个中断源。

5.2 中断控制

51 单片机内部有一个中断允许寄存器 IE（A8H），它控制着每一个中断允许或屏蔽，其格式如下：

	D_7			D_4	D_3	D_2	D_1	D_0
IE	EA	/	/	ES	ET1	EX1	ET0	EX0
位地址	AFH			ACH	ABH	AAH	A9H	A8H

1．EA 为 CPU 的中断开放标志位，EA＝ 1 ，CPU 开放中断，EA＝ 0 ，CPU 屏蔽所有中断。

2．ES 为串行口中断允许位，ES=1，允许串行口中断，ES=0，禁止串行口中断。

3．ET1 为定时/计数器 T1 的溢出中断允许位，ET1=1，允许 T1 中断，ET1=0，禁止 T1 中断。

4．EX1 为外部中断 1 的中断允许位，EX1=1，允许外部中断 1 中断，EX1=0，禁止外部中断 1 中断。

5．ET0 为定时/计数器 T0 的溢出中断允许位，ET0=1，允许 T0 中断，ET0=0，禁止 T0 中断。

6．EX0 为外部中断 0 的中断允许位，EX0=1，允许外部中断 0 中断，EX0=0，禁止外部中断 0 中断。

IE 复位值为 0，必需由用户程序自己设置，以下是一个中断的例子，

```
        ORG     0000H
        JMP     MAIN
        ORG     0100H
MAIN:
        SET     EA          ；CPU 中断允许
        SET     ES          ；串行口中断允许
        CLR     ES          ；串行口中断禁止

        END
```

5.3 中断优先级

51 单片机有两个中断优先级，对每个中断请求可通过设置，改变其优先级别，高或低，并能实现两级嵌套。优先级的改变是通过对 IP 进行设置的，IP 的格式如下：

	D_7			D_4	D_3	D_2	D_1	D_0
IP	/	/	/	PS	PT1	PX1	PT0	PX0
位地址				BCH	BBH	BAH	B9H	B8H

1. PS：串行口中断优先级控制位。PS=1，串行口中断定义为高优先级中断；PS=0，定义为低优先级。
2. PT1：定时器 T1 中断优先级控制位。PT1=1，定时器中断定义为高优先级中断；PT1=0，定义为低优先级。
3. PX1：外部中断 1 中断优先级控制位。PX1=1，外部中断 1 定义为高优先级中断；PX1=0，定义为低优先级。
4. PT0：定时器 T0 中断优先级控制位。PT0=1，定时器中断定义为高优先级中断；PT0=0，定义为低优先级。
5. PX0：外部中断 0 中断优先级控制位。PX0=1，外部中断 0 定义为高优先级中断；PX0=0，定义为低优先级。

中断优先级控制寄存器 IP 的各位都由用户置位和复位，IP 复位值为 00H ，各个中断源均为低优先级中断。

同时收到几个同一级的中断时，CPU 的响应顺序如下图：

中断源	中断级别
外部中断 0	最高
T0 溢出中断	|
外部中断 1	|
T1 溢出中断	↓
串行口中断	最低

如果将 IP 设置为 IP＝10H，则串行口中断将最先响应，其它中断仍按上述顺序。

* 5.4 SST89 系列中断控制

SST89C54/SST89C58 提供 6 个中断源，2 个中断优先级，SST89E58RD/V554RC 和 SST89E516RD/V564RD 提供 9 个中断源，4 个中断优先级。

上面提到了 2 个中断优先级响应的顺序，如果设置了 4 个优先级，譬如两个中断优先级寄存器设置如下：

Interrupt Priority (IP)

Location	7	6	5	4	3	2	1	0
B8H	-	PPC	PT2	PS	PT1	PX1	PT0	PX0
		0	0	1	1	0	0	0

Interrupt Priority High (IPH)

Location	7	6	5	4	3	2	1	0
B7H	-	PPCH	PT2H	PSH	PT1H	PX1H	PT0H	PX0H
		0	1	1	0	0	0	0

如果同时有 PCA、定时器 T2、串口、定时器 T1 四个中断触发，那么它们的响应顺序应该是串口、定时器 T2、定时器 T1、PCA。

5.5 外 部 中 断

这一节主要讲叙外部中断 INT0,INT1 的操作，对于定时器，串行口中断将劈出单独章节进行详细讲叙。

外部中断的控制与定时器/计数器的控制共用同一个寄存器 TCON，TCON 格式如下：

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
TCON	TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
位地址	8FH		8DH		8BH	8AH	89H	88H

IE1 外部中断 1 的请求标志位。当检测到外部中断引脚 1（P3.3）上存在有效中断请求信号时，由硬件使 IE1 置 1。当 CPU 响应该中断请求后，由硬件使 IE1 清 0。

IT1 外部中断 1 的中断触发方式控制位。IT1=0 时，外部中断 1 为电平触发方式，当有效请求信号为低电平时 IE1 置 1。IT1=1 时，外部中断 1 为边沿触发方式，有效请求信号为下降沿。

各控制位的说明：

1. IE1——外部中断 1 的请求标志位。当检测到外部中断引脚 1（P3.3）上存在有效中断请求信号时，由硬件使 IE1 置 1。当 CPU 响应该中断请求后，由硬件使 IE1 清 0。
2. IT1——外部中断 1 的中断触发方式控制位。

IT1=0 时，外部中断 1 为电平触发方式，当有效请求信号为低电平时 IE1 置 1。IT1=1 时，外部中断 1 为边沿触发方式，有效请求信号为下降沿。

※IE0,IT0 完成对外部中断 0 的控制，其控制方式与 IT1,IE1 相同。

例：假设允许外部中断 0，电平触发方式，且禁止其它中断。试根据条件写出其汇编程序。

解：1.用字节操作指令

```
MOV     IE    ,   #81H            ；开中断
MOV     TCON  ,   #00H            ；IT1=0，电平触发
```

2. 位操作指令

```
CLR     IT0                       ；外部中断 0 设为电平触发
SET     EX0                       ；外部中断 0 允许
SET     EA                        ；中断允许
```

第 六 章　定时器/计数器

51 单片机内部设置两个 16 位可编程的定时器/计数器 T0 和 T1，它们具有计数器方式和定时器方式两种工作方式及四种工作模式。

6.1 定时器概述

两个定时/计数器都有定时和计数的功能，它们可用于定时控制、延时、对外部事件计数和检测等场合。

定时器 T0、T1 的结构及与 CPU 的关系如图 6-1 所示。两个 16 位定时器实际上都是 16 位加 1 计数器。其中，T0 由 2 个 8 位特殊功能寄存器 TH0 和 TL0 构成；T1 由 TH1 和 TL1 构成。每个定时器都可由软件设置为定时工作方式或计数工作方式。这些功能都由特殊功能寄存器（SFR）TMOD 和 TCON 所控制。

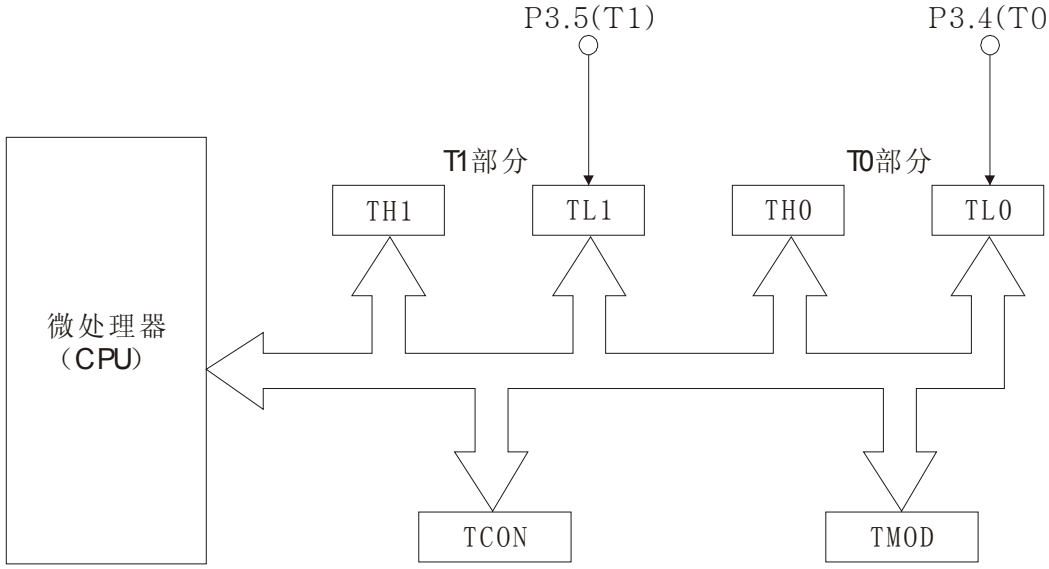


图 6-1 定时器与 CPU 的关系

当 8031 内部的定时器/计数器被选择为定时工作方式时，计数输入信号是内部时钟脉冲，每个机器周期产生一个脉冲使计数器加 1。因此，定时器/计数器的输入脉冲周期与机器周期一样，为振荡频率的 1/12。当采用 12MHz 频率的晶体时计数速率为 1MHz，输入脉冲的周期间隔为 1 微秒。由于定时的精度决定于脉冲的周期，因此，当需要高精度的定时器时，应尽量选择频率较高的晶体。

当设置成计数工作方式时，通过引脚 T0（P3.4）和 T1（P3.5）对外部脉冲计数。当输入信号产生由 1 至 0 的跳变时，计数器的值增 1。在每个机器周期的 S5P2 期间，对外部输入进行采样。如在第一个周期采得的值为 1,而在下一个周期中采得的值为 0,则在紧跟着的再下一个周期中的 S3P1 期间，计数器加 1。由于确认一次下跳变要花两个机器周期，即 24 个振荡周期，因此外部输入的计数脉冲的最高频率为振荡频率的 1/24。例如选用 6MHz 频率的晶体允许输入的脉冲频率为 250kHz，如果选用 12MHz 频率的晶体，则可输入 500kHz 的外部脉冲。

对外部输入信号的占空比并没什么限制，但为了确保某一给定的电平在变化之前至少被采样一次，则这一电平至少要保持一个机器周期，故对信号的基本要求如图 6－2 所示：

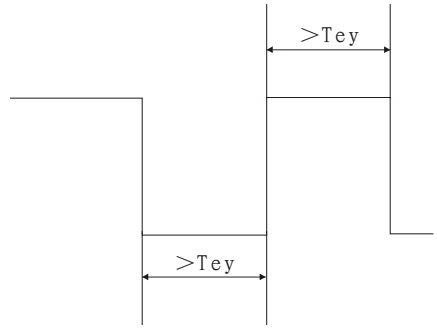


图 6－2

不管是定时还是计数工作方式，定时器 T0 或 T1 在对内部时钟或外部事件计数时，不影响 CPU 工作，直到计数溢出才产生中断。

除了可以选择定时或计数工作方式外，每个定时器/计数器还有四种工作方式可选，我们会在后几节中详细讲述。

6.2 定时器控制寄存器

定时器有两个控制字，TMOD 和 TCON，用软件对它们进行设置可以控制 T0 和 T1 的工作模式和操作功能。其复位值为 0。

6.2.1 工作方式控制寄存器 TMOD

TMOD 用于 T0 和 T1 的工作方式及四种工作模式，其各位的定义格式如下面表所示：

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
GATE	$C/\overline{T}$	M1	M0	GATE	$C/\overline{T}$	M1	M0
定时器T1方式字段				定时器T0方式字段			

其中，低四位用于定时器 T0，高四位用于定时器 T1。

M1 和 M0 工作方式选择位

M0 M1 可形成四种编码，每个编码对应一种情况，其对应操作模式如下表：

M1	M0	功能说明
0	0	方式0，为13位的定时器／计数器
0	1	方式1，为16位的定时器／计数器
1	0	方式2为常数自动重新装入的8位定时器／计数器
1	1	仅适用于T0，分为二个8位计数器，对T1停止计数

C/T 定时器和外部计数方式选择

C/T＝0 为定时器方式。定时器的计数脉冲来自晶振的 1/12 分频。

C/T＝1 为计数器方式。计数器的计数脉冲来自 T0（P3.4）或 T1(P3.5)的外部脉冲。

GATE 门控位

GATE 控制定时/计数器的计数是否受外部引脚的控制。GATE＝1 时，定时/计数器的计数受外部引脚输入电平的控制（INT0 控制 T0，INT1 控制 T1），只有当这两个引脚为高电平且由软件使 TR0（或 TR1）置 1 时，才能启动定时器。GATE＝0 时，定时/计数器的工作不受外部引脚的控制。

TMOD 各位定义及具体意义可以见下图

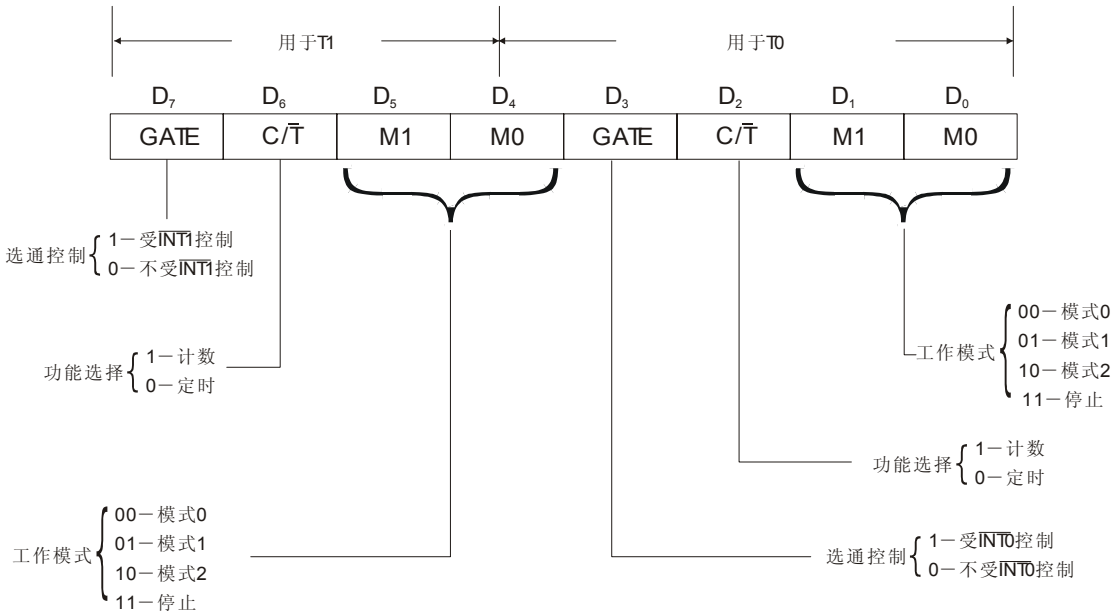


图 6-3 TMOD 各位定义

6.2.2 控制寄存器 TCON

定时器控制寄存器 TCON 字节地址 88H，可位寻址，位地址 88H～8FH，各位定义格式如下：

TCON

D_7	D_6	D_5	D_4	D_3	D_0
TF1	TR1	TF0	TR0		

TCON 的低 4 位控制外部中断，我们已经在前一章中讲述过了。

TCON 的高 4 位控制位的含义如下：

1. TR0——定时器 T0 的运行控制位

可通过软件置 1 或清 0 来关闭或启动 T1。如果 GATE＝1,则需要 TR0＝1 且 INT0 为高时才允许 T0 对外部进行计数。

2. TF0——定时器 T0 溢出标志位

当 T0 计数溢出，由硬件对 TF0 置位，当中断响应结束由硬件对 TF0 清 0,该位也可由软件清 0。

3. TR1——定时器 T1 的运行控制位

可通过软件置 1 或清 0 来关闭或启动 T1。如果 GATE＝1,则需要 TR1＝1 且 INT0 为高时才允许 T0 对外部进行计数。

4. TF1——定时器 T1 溢出标志位

当 T1 计数溢出，由硬件对 TF1 置位，当中断响应结束由硬件对 TF1 清 0,该位也可由软件清 0。

下图是对 TCON 各控制位的描述：

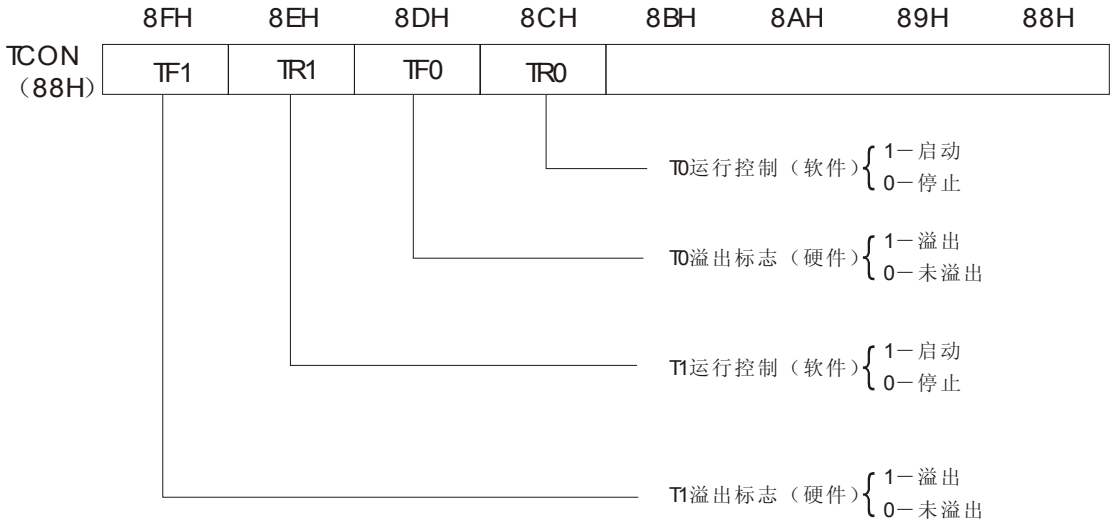


图 6-4　TCON 各位定义

6.3 定时器的四种工作方式

6.3.1　工作方式 0

当 M1　M0 为 00 时，定时器/计数器被选为工作模式 0,这时定时器/计数器的等效逻辑结构如图所示（以定时器/计数器 T1 为例，TMOD.5TMOD.4＝00）。

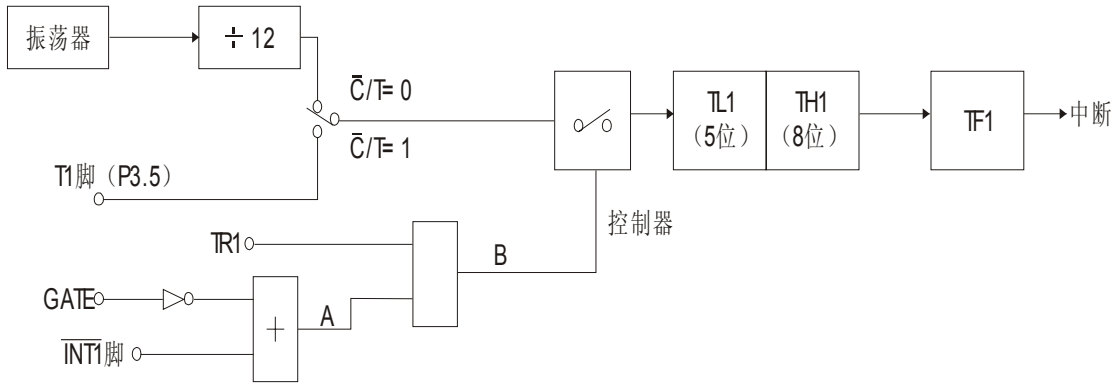


图 6-5 T1 模式 0 逻辑结构

在这种模式下，16 位寄存器（TH0 和 TL0）只用了 13 位。其中，TL0 的高 3 位未用，其余位为整个 13 位的低 5 位，TH0 占高 8 位。当 TL0 的低 5 位溢出时，向 TH0 进位；TH0 溢出时，向中断标志位 TF0 进位（硬件置位 TF0），并申请中断。T0 是否溢出可查询 TF0 是否被置位，以产生 T0 中断。

在图 6-5 中，C/T＝0 时，控制开关接通振荡器 12 分频输出端，T0 对机器周期计数，这就是定时工作方式。其定时时间为

$$t = (2^{13} - \text{T0 初值}) \times \text{振荡周期} \times 12$$

当 C/T＝1 时，控制开关使引脚 T0（P3.4）与 13 位计数器相连，外部计数脉冲由引脚 T0（P3.4）输入，当外部信号电平发生由 1 到 0 跳变时，计数器加 1。这时，T0 成为外部事件计数器，这就是计数工作方式。

GATE＝0 时，使“或”门输出 A 点电位保持为 1,“或”门被锁。于是，引脚 INT0 输入信号无效。这时，“或”门输出的 1 打开“与”门。B 点电位取决于 TR0 的状态，于是，由 TR0 一位就可控制计数开关开启或关断 T0。若软件使 TR0 置 1,便接通计数开关，启动 T0 在原值上加 1 计数，直到溢出。溢出时，13 位寄存器清 0,TF0 置位，并申请中断，T0 从 0 重新开始计数。若 TR0＝0,则关断计数开关，停止计数。

当 GATE＝1 时，A 点电位取决于 INT0（P3.2）引脚的输入电平。仅当 INT0 输入高电平且 TR0＝1 时，B 点才是高电平，计数开关闭合，T0 开始计数；当 INT0 由 1 变 0 时，T0 停止计数。这一特性可以用来测量在 INT0 端出现的正脉冲宽度。

例：假设时钟频率采用 6MHz，要在 P1.0 上输出一个周期为 2ms 的方波，方波的周期用定时器 T0 来确定，采用中断方法来实现。

解：

根据题意我们可以在 T0 中设置一个时间常数，使其每隔 1ms 产生一次中断，CPU 响应中断后，在中断服务程序中对 P1.0 取非。中断入口 T0 地址为 000BH。为此来做如下几步：

1. 确定定时常数

$$\text{机器周期}=\frac{12}{\text{晶振频率}}\times\frac{12}{6\times10^{6}}=2\text{us}$$

设：需要初值为 X，则

$$(2^{13}-X)\,2\times10^{-6}=1\times10^{-3}$$

X＝7692

化为 16 进制 X＝1E0CH

根据 13 位定时器特性，初值应为：

TH0＝0F0H

TL0＝0CH

2. 设计初始化程序

包括定时器初始化和中断系统初始化，主要是对 IP、IE、TCON、TMOD 的相应位进行正确的设置，并将时间常数送入定时器中。在本例中，假设系统从复位开始，TMOD、TCON 均为 00H，因此不必对 TMOD 操作。

3. 设计中断服务程序和主程序

中断服务程序除了完成产生要求的方波外，还要注意将时间常数重新送入定时器中，为下次产生中断作准备。主程序可以完成任何其他的工作，一般情况下常常是键盘和显示程序。在本例中，用一条转自自身的短跳转指令来代替主程序。

按上面的设计程序清单如下：

```
        ORG     0000H
        AJMP    MAIN            ；转主程序
        ORG     000BH
        AJMP    ITOP            ；转中断入口程序

        ORG     0100H
MAIN:
        MOV     SP,#60H
        ACALL   PTOMO
        HEHE:
        AJMP    HERE
        PTOMO:
        MOV     TL0,#0CH        ；T0 置初值
        MOV     TH0,#0F0H
```

```
        SETB    TR0             ；启动 T0
        SETB    ET0             ；允许 T0 中断
        SETB    EA              ；CPU 开放中断
        RET
ITOP:
        MOV     TL0,#0CH        ；T0 置初值
        MOV     TH0,#0F0H
        CPL     P1.0            ；P1.0 位取反
        RETI
```

6.3.2 工作方式 1

该模式对应的是一个 16 位的定时器/计数器，其结构与操作几乎与模式 0 完全相同，唯一的差别是：在模式 1 中，寄存器 TH0 和 TL0 是以全部 16 位参与操作。用于定时工作方式时，定时时间为

$$t = （2^{16} - T0\ 初值）\times 振荡周期 \times 12$$

用于计数工作方式时，计数长度为 216=65536 个外部脉冲。
其工作结构图如下：

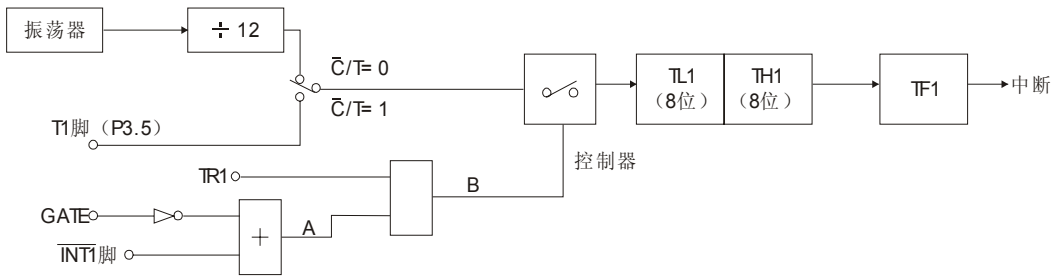


图 6-6 T1 模式 1 逻辑结构

例：假设利用定时器 T0 模式 1 产生一个 50Hz 的方波，由 P1.0 输出，采用 12MHz 时钟，并假定 CPU 不作其它工作，因而可以采用查询的方式进行控制。

初值可由下式算得

$$（216-X）\times 50 = 1 \times 10^6$$

因而 X＝45536＝0B1E0H。程序如下：

```
        MOV   TMOD，＃01H ；设置定时器 0 模式 1
        SETB  TR0
LOOP：
        MOV   TH0，＃0B1H
        MOV   TL0，＃0E0H
LOOP1：
        JNB   TF0，LOOP1
        CLR   TF0
        CPL   P1.0
        SJMP  LOOP
```

6.3.3 工作方式 2

方式 2 把 TL0(或 TL1)配置成一个可以自动重装载的 8 位定时器/计数器，结构图如下所示。

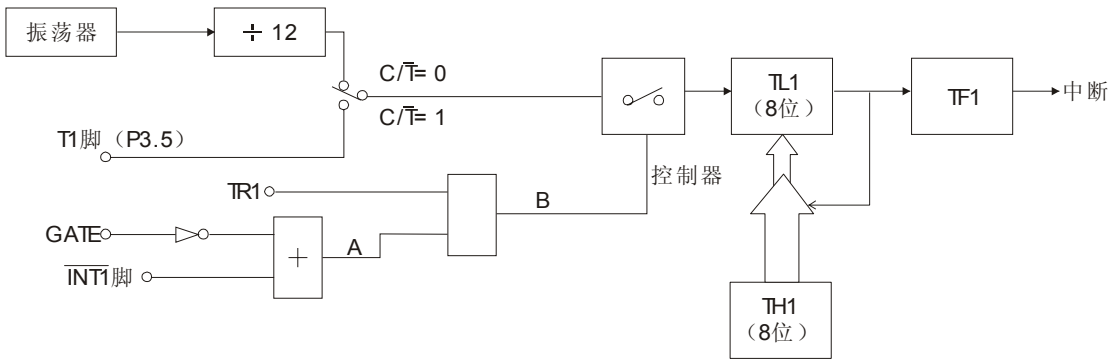


图 6-6 T1 模式 2 逻辑结构

TL0 计数溢出时，不仅使溢出中断标志位 TF0 置 1,而且还自动把 TH0 中的内容重新装入到 TL0 中。这里，16 位计数器被拆成 2 个，TL0 用于 8 位计数器，TH0 用于保存初值。

在程序初始化时，TL0 和 TH0 由软件赋予相同的初值。一旦 TL0 计数溢出，便置位 TF0，并将 TH0 中的初值再自动装入 TL0,继续计数，循环重复。用于定时工作方式时，其定时时间（TF0 溢出周期）为

$$t = (2^8 - \text{T0 初值}) \times \text{振荡周期} \times 12$$

用于计数工作方式时，最大计数脉冲个数为 2^8=256 个外部脉冲。

这种工作方式可省去用户软件中重装常数的语句，并可产生相当精确的定时时间，特别适用于作串行口波特率发生器。

例：把 T0（P3.4）作为外部中断请求输入线，即 T0 引脚产生负跳变时，向 CPU 请求中断。下面的程序将 T0 定义为模式 2 计数方式，计数器初值为 FFH，即计数输入端 T0（P3.4）发生一次负跳变时，计数器加 1 即产生溢出标志，向 CPU 发中断。程序在 T0 产生一次负跳变后，使 P1.0 产生 2ms 的方波。其中定时器 T1 用于产生 1ms 定时（6MHz）。

```
        ORG     0000H
RESET:
        AJMP    MAIN            ；复位入口转主程序
        ORG     000BH
        AJMP    ITOP            ；转 T0 中断服务程序
        ORG     001BH
        AJMP    IT1P            ；转 T1 中断服务程序
        ORG     0100H
MAIN:
        MOV     SP,#60H
        ACALL   PTOM2           ；对 T0、T1 初始化
LOOP:
        MOV     C,P1.1          ；T0 产生过中断了吗？
        JNC     LOOP
        SETB    TR1             ；启动 T1
        SETB    ET1             ；允许 T1 中断
HERE:
        AJMP    HERE
PTOM2:
        MOV     TMOD,#16H       ；T0 初始化程序
        MOV     TL0,#0FFH       ；T0 置初值
        MOV     TH0,#0FFH;
        SETB    TR0             ；启动 T0
        SETB    ET0             ；允许 T0 中断
        MOV     TL1,#0CH        ；T1 置初值
        MOV     TH1,#0FEH
        CLR     P1.1
        SETB    EA              ；CPU 开放中断
        RET
ITOP:
        CLR     TR0             ；停止 T0 计数
```

```
                SETB    P1.1                ；建立标志
                RETI
IT1P:
                MOV     TL1,#0CH
                MOV     TH1,#FEH
                CPL     P1.0                ；P1.0 位取反
                RETI
```

6.3.4 工作方式 3

工作方式 3 对 T0 和 T1 大不相同。

若将 T0 设置为模式 3,TL0 和 TH0 被分成两个相互独立的 8 位计数器，如图所示

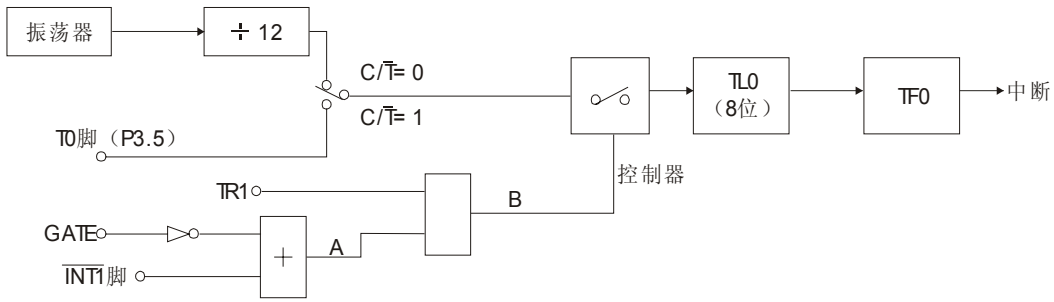


图 6-6 T1 模式 3 逻辑结构

模式 3 是为了增加一个附加的 8 位定时器/计数器而提供的，使 51 单片机具有 3 个定时器/计数器。模式 3 只适用于定时器/计数器 T0，定时器 T1 处于模式 3 时相当于 TR1＝0,停止计数（此时 T1 可用来做串行口波特率发生器）。

上图中，TL0 的各控制位、引脚和中断源，即 C/T、GATE、TR0、TF0 和 T0（P3.4）引脚，INT0 引脚。TL0 除仅用 8 位寄存器外，其功能和操作与模式 0(13 位计数器)、模式 1(16 位计数器)完全相同。TL0 也可工作在定时器或计数器方式。

一般情况下，当定时器 T1 用作串行口的地波特率发生器时，定时器/计数器 T0 才工作在方式 3。当定时器 T0 处在工作方式 3 时，定时器 T1 可定为方式 0,方式 1 和方式 2,作为串行口的波特率发生器，或不需要中断地场合。T1 没有工作方式 3,如果将 T1 设置为方式 3 ，就会使 T1 停止计数。

例：下面的程序使定时器／计数器 T0 工作在模式 3，TL0 和 TH0 作为两个独立的 8 位定时器／计数器。分别产生 200 微秒和 400 微秒的定时中断，使 P1.0 和 P1.1 产生 400 微

秒和 800 微秒的方波（晶振频率为 6MHz）。

```
                ORG         0000H
RESET:
                AJMP        MAIN            ；复位入口转主程序
                ORG         000BH
                AJMP        ITOP            ；转 T0 中断服务程序
                ORG         001BH
                AJMP        IT1P            ；转 T1 中断服务程序
                ORG         0100H
MAIN:
                MOV         SP,#60H
                ACALL       PTOM3           ；对 T0、T1 初始化
HERE:
                AJMP        HERE
PTOM3:
                MOV         TMOD,#03H       ；T0 初始化程序
                MOV         TL0,#9CH        ；T0 置初值
                MOV         TH0,#038H
                SETB        TR0             ；启动 T0
                SETB        ET0             ；允许 T0 中断
                SETB        TR1             ；启动 T0
                SETB        ET1             ；允许 T0 中断
                SETB        EA              ；CPU 开放中断
                RET
ITOP:
                MOV         TL0,#9CH
                CPL         P1.1
                RETI
IT1P:
                MOV         TH0,#38H
                CPL         P1.0            ；P1.1 位取反
                RETI
```

6.3.5 定时器/计数器的综合应用

为了让大家熟悉定时器计数器的使用，我们给出一个综合应用的例子，希望能对大家有所帮助。这个程序是用 C 语言写的，如果大家对 C 语言不熟悉，可以去参阅 C 语言

的书籍。

例：试用定时器产生一个任意频率且占空比可调的波形。

```
/*=====================================================*/
/*  频率为        :  f                                 */
/*  占空比        :  m:n    m 为正                     */
/*  脉冲个数      :  counters                          */
/*  设定的脉冲个数：geshu0                             */
/*=====================================================*/
void pulse( )
{
      long   int m , n;
      m = 22118400 / ( 12 * f0 ) ;
      n = m * percent / 100 ;

      Y0 = 65567 - n ;            /*Y0 为正*/
      Y1 = 65567 - m + n ;      /*这两个初值是在试验中得出的，不同的系统就不一样*/

      TMOD = 0x21 ;               /*定时器 0，方式 1*/
      TH0 = Y0 / 256 ;
      TL0 =   Y0 % 256 ;
      EA = 1 ;
      ET0 = 1 ;
      TR0 = 1 ;
      signal=0;

      return ;
 }

 timer0( ) interrupt 1 using 1
 {
    if(signal==1)
    {
        TH0 = Y1 / 256 ;
        TL0 = Y1 % 256 ;

     }
     else
     {
```

```
        TH0 = Y0 / 256 ;
        TL0 = Y0 % 256 ;
    }
    signal = ! signal ;
    EA = 1 ;
    ET0 = 1 ;
    TR0 = 1 ;

    Counters --;
    if(counters==0)
   {
        signal = 0;
        EA = 1;
        ET0 = 0;
        TR0 = 0;
        return ;
    }
}
```

这个程序是在实际系统中应用过的，系统的晶振是 11.0592MHz，在产生 10Hz－250Hz 且占空比在 1：1-1：99 的方波时，精确度可达 0.01Hz。

以上程序如果用汇编实现起来，将会比较复杂，下面给出用汇编写的定时器 T0 的中断服务程序：

```
    TIMER0:
            MOV     A , # 1
            CJNE    A , SIGNAL , TIMER0_2
    TIMER0_1:
            MOV     A , # Y1
            MOV     B , # 256
            DIV     AB
            MOV     TH0 , A
            MOV     TL0 , B
    TIMER0_2:
            MOV     A , # Y0
            MOV     B , # 256
            DIV     AB
            MOV     TH0 , A
            MOV     TL0 , B

            MOV     A , SIGNAL
```

```
                CPL     A
                SET     EA
                SET     ET0
                SET     TR0

                DEC     COUNTER

                MOV     A , #0
                CJNE    A , COUNTER , EXIT
TIMER0_3：
                MOV     SIGNAL , A
                SET     EA
                SET     ET0
                SET     TR0

EXIT：          RETI
```

第 七 章　串行通信接口技术

串行通信是微机接口的一个重要组成部分，有着极其广泛的应用。随着微机特别是单片机的发展，其应用已从单机逐渐转向多机或联网，而多机应用的关键又在微机通信。微机通信有串行和并行两种通信方式，并行通信可以提高数据交换速度而串行通信可以节省系统资源，降低系统成本。串行通信又分为同步串行通信和异步串行通信。本章将就串行通信进行详细讲述，末了还给出应用实例，力求反应目前串行通信的新技术和新发展。

7.1　串行通信的概念

7.1.1　并行通信和串行通信

在微机系统中，CPU 与外部的基本通信方式有两种：

并行通信——数据各位同时传送；

串行通信——数据一位位顺序传送。

图 7-1 是这两种方式的示意图。一般快速设备之间采用并行通信，譬如 CPU 与存储设备、存储器与存储器、主机与打印机等都采用并行通讯。并行通讯，有多少位数据就必须有多少根数据线，如下图是 11 位数据就有 11 根数据线。串行通信最少可以只需一根通信线，只发或只收。因而大大节省了系统资源，降低了系统成本。由于只用一根数据线，所以是以降低传送速度来换取资源的，它常用在传送距离远，速度要求不高的场合。

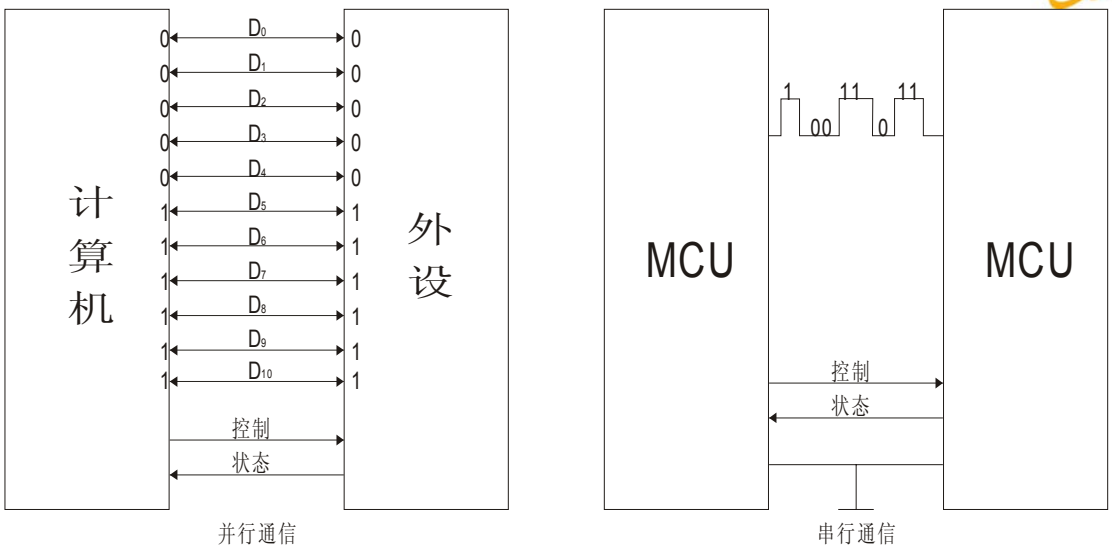
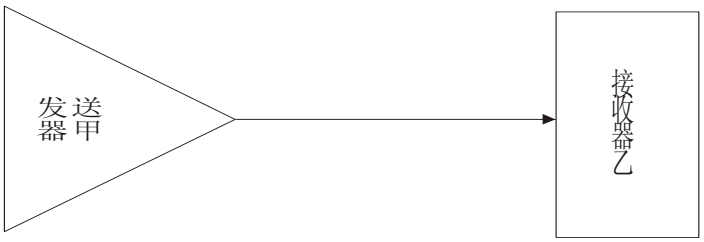


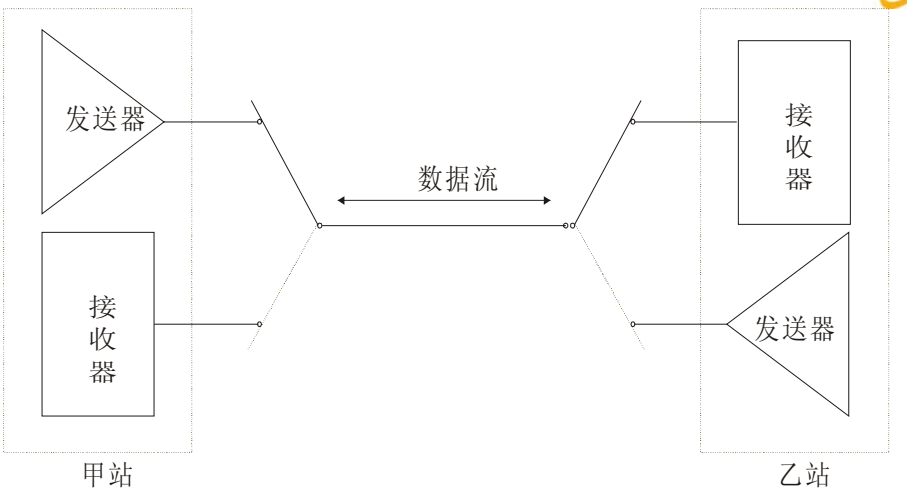
图 7-1　并行通信与串行通信

7.1.2　串行通信的传送方式

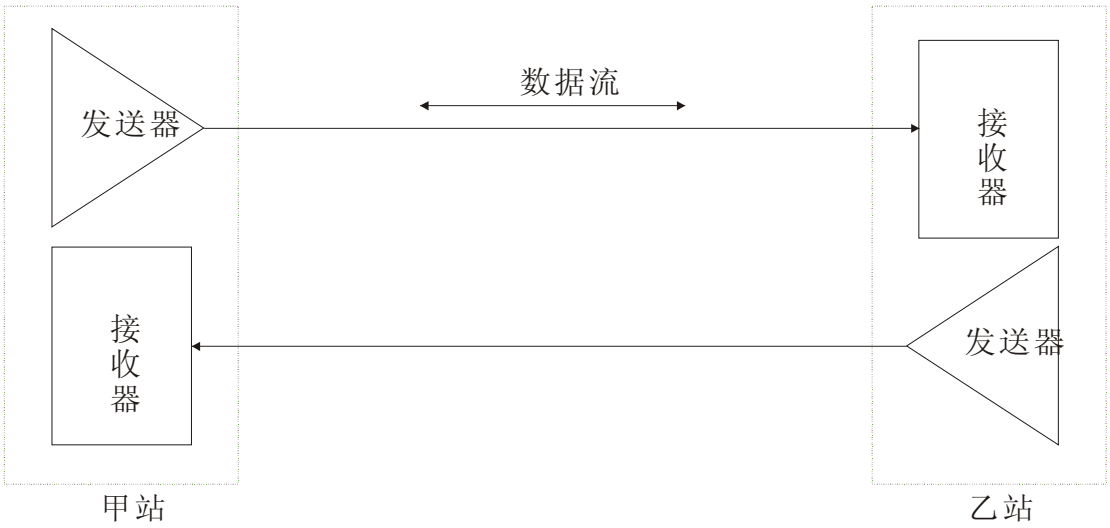
串行通信的传送方式通常有 3 种：一种为单向（或单工）配置，只允许数据向一个传送；另一种是半双工配置，允许数据向两个方向中的任一方向传送，但每次只能有一个站发送；第三种传送方式是全双工配置，允许同时双向传送数据，因此，全双工配置是一对单向配置，它要求两端的通信设备具有完整和独立的发送和接收能力。图 7-2 所示为串行通信中的数据传送方式。



（a） 单工方式



（b）半双工方式



（c）全双工方式

图 7-2　串行通信传输方式

7.1.3　异步通信和同步通信

串行通信进行数据传送时是将要传送的数据按二进制位，依据一定的顺序逐位发送到接收方。其有两种通信方式：

1、 异步通信

异步通信是我们最常采用的通信方式，我们后面的例子都是采用的异步通信方式。

异步通信采用固定的通信格式，数据以相同的帧格式传送。如图 7-3 所示，每一帧由起

始位、数据位、奇偶校验位和停止位组成。

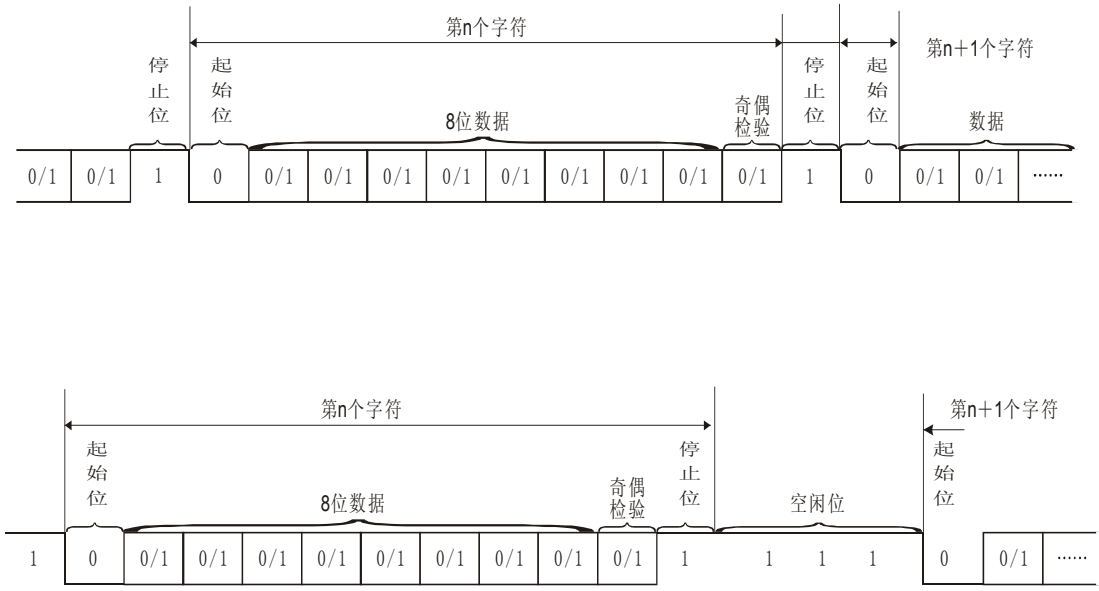


图 7－3　异步通信的一帧数据格式

在通信线上没有数据传送时处于逻辑“1”状态。当发送设备发送一个字符数据时，首先发出一个逻辑“0”信号，这个逻辑低电平就是起始位。起始位通过通信线传向接收设备，当接收设备检测到这个逻辑低电平后，就开始准备接收数据信号。因此，起始位所起的作用就是表示字符传送开始。

起始位后面紧接着的是数据位，它可以是 5 位、6 位、7 位、或 8 位。数据传送时，低位在前。

奇偶校验位用于数据传送过程中的数据检错，数据通信时通信双方必须约定一致的奇偶校验方式。就数据传送而言，奇偶校验位是冗余位，但它表示数据的一种性质。也有的不要校验位。

在奇偶校验位或数据位后紧接的是停止位，停止位可以是一位、也可以是 1.5 位或 2 位。接收端收到停止位后，知道上一字符已传送完毕，同时，也为接收下一字符作好准备。若停止位后不是紧接着传送下一个字符，则让线路保持为“1”。“1”表示空闲位，线路处于等待状态。存在空闲位是异步通信的特性之一。

2、 同步通信

同步通信时，通信双方共用一个时钟，这是同步通信区分于异步通信的最显著的特点。在异步通信中，每个字符要用起始位和停止位作为字符开始和结束的标志，以致占用了时间。

所以在数据块传送时，为提高通信速度，常去掉这些标志，而采用同步通信。同步通信中，数据开始传送前用同步字符来指示（常约定 1～2 个），并由时钟来实现发送端和接收端的同步，即检测到规定的同步字符后，下面就连续按顺序传送数据，直到一块数据传送完毕。同步传送时，字符之间没有间隙，也不要起始位和停止位，仅在数据开始时用同步字符 SYNC 来指示，其数据格式见图 7-4。

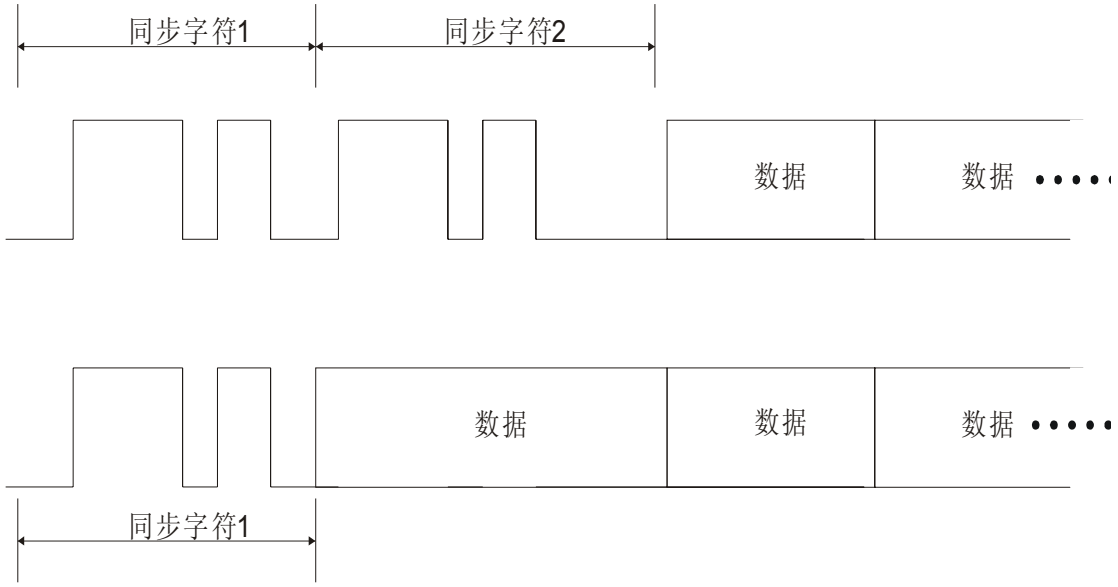


图 7－4　同步传送的数据格式

同步通信和异步通信相比有以下特点：

1. 以同步字符作为传送的开始，从而使收发双方取得同步。
2. 每位占用的时间相等。
3. 字符数据之间不允许有空位，当线路空闲或没字符可发时，发送同步字符。

同步字符的插入可以是单同步字符或双同步字符，如图 7-4 所示同步字符也可以由用户约定，当然也可以采用 ASCII 码中规定的 SYN 代码，即 16H。

在同步传送时，要求用时钟来实现发送端和接收端之间的同步。为了保证接收正确无误，发送方除了传送数据外，还要传送同步时钟。

同步通信虽然可以提高传送速度，可达 56Kb/s 或更高，但实现起来颇为复杂，因此实际较少使用。

7.1.4　波特率和接收发送时钟

1. 波特率（Baud rate）

波特率是指数据传送时，每秒传送数据二进制代码的位数，它的单位是位/秒（b/s）。1 波特就是一位每秒。假设数据传送速率是每秒 120 字符，而每个字符格式包括 10 个代码位（1 个起始位、一个终止位、8 个数据位），这时传送的波特率为：

$$10 \times 120 = 1200\text{b/s}$$

位传送时间宽度 T_d＝波特率的倒数，则上式中的 T_d＝1/1200s＝0.883ms。

在异步串行通信中，接收设备和发送设备保持相同的传送波特率，并以每个字符数据的起始位与发送设备保持同步。起始位。数据位。奇偶位和停止位的约定，在同一次传送过程中必须保持一致，这样才能成功的传送数据。

2.接收/发送时钟

二进制数据系列在串行传送过程中以数字信号波形的形式出现。不论接收还是发送，都必须有时钟信号对传送的数据进行定位。接收/发送时钟就是用来控制通信设备接收/发送字符数据速度的，该时钟信号通常由外部时钟电路产生。

在发送数据时，发送器在发送时钟的下降沿将移位寄存器的数据串行移位输出；在接收数据时，接收器在接收时钟的上升沿对接收数据采样，进行数据位检测，如图 7-5 所示。

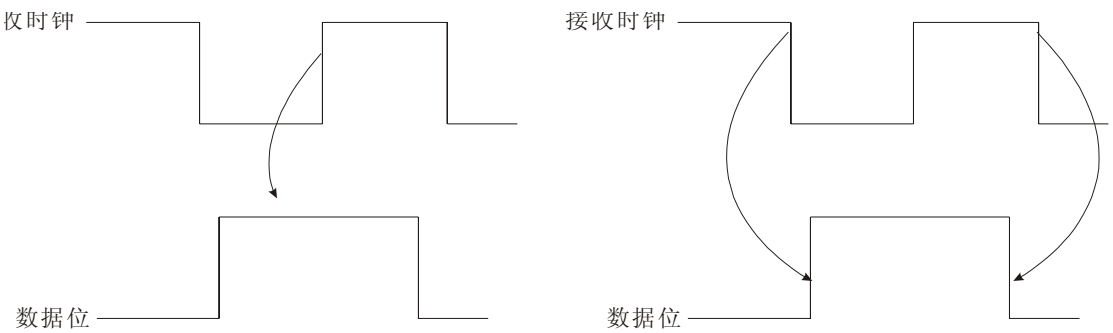


图 7－5　接收时钟和发送时钟

接收/发送时钟频率与波特率有如下关系：

收/发时钟频率 ＝ n × 收/发波特率

n＝1，16，64

在同步传送方式，必须取 n＝1，即接收/发送时钟的频率等于收/发波特率。在异步传送方式，n＝1,16,64,即可以选择接收/发送时钟频率是波特率的 1,16,64 倍。因此可由要求的传送波特率及所选择的倍数 n 来确定接收/发送时钟的频率。

例如，若要求数据传送的波特率为 300Baud，则

接收/发送时钟频率＝300Hz （n＝1）
接收/发送时钟频率＝4800Hz （n＝16）
接收/发送时钟频率＝19.2kHz （n＝64）

接收/发送时钟的周期 Tc 与传送的数据位宽之间的关系是：

$$Tc = Td / n$$

若取 n＝16,那么异步传送接收数据实现同步的过程如下：接收器在每一个接收时钟的上升沿采样接收数据线，当发现接收数据线出现低电平时就认为是起始位的开始，以后若在连续撤 8 个时钟周期（因 n＝16,故 Td＝16Tc）内检测到接收数据线仍保持低电平，则确定它为起始位（不是干扰信号）。通过这种方法，不仅能够排除接收线上的噪声干扰，识别假起始位，而且能够相当精确的确定起始位的中间点，从而提供一个正确的时间基准。从这个基准算起，每隔 16Tc 采样一次数据线，作为输入数据。一般来说，从接收数据线检测到一个下降沿开始，若其低电平能保持 n/2Tc（半位时间），则确定为起始位，其后每隔 nTc 时间（一个数据时间）在每个数据位的中间点采样。

由此可见，接收/发送时钟对于收/发双方之间的数据传输达到同步是至关重要的。

7.2 串行通信总线标准及接口

采用标准接口后，能很方便的把各种计算机、外部设备、测量仪器有机的连接起来，构成一个测量、控制系统。串行通信总线标准实际上是指异步串行通信的接口，目前这类接口有四类：

RS－232C（RS－232A，RS－232B）
RS－449,RS－422,RS－423 和 RS－485
20mA 电流环
USB 接口

但工业上常用的接口只有 RS－232C，RS－422，RS－485 等几种，USB 是最新出现的接口，常用于连接电脑外设。

7.2.1 RS－232C 接口

EIA RS－232C 是美国电子工业协会正式公布的串行总线标准，也是目前最常用的串行

接口标准，用来实现计算机与计算机之间、计算机与外设之间的数据通信。RS－232C 串行接口总线适用于：设备之间的通信距离不大于 15m，传输速率最大为 20kB/s。

1、接口信号

一个完整的 RS－232C 接口有 22 根线，采用标准 25D 插头。表 7-1 给出了 RS－232C 串行标准接口信号的定义以及信号的分类。

表 7-1 RS－232 接口信号

引脚号	缩写符	信号方向	说　　明
1			屏蔽（保护）地
2	TXD	从终端到调制解调器	发送数据
3	RXD	从调制解调器到终端	接收数据
4	RTS	从终端到调制解调器	请求发送
5	CTS	从调制解调器到终端	清除发送
6	DSR	从调制解调器到终端	数据装置就绪
7		———	信号地
8	DCD	从调制解调器到终端	接收线信号检出（载波检测）
9	—	———	保留供测试用
10	—	———	保留供测试用
11	—	———	未定义
12	DCD	从调制解调器到终端	辅信道接收线信号检测
13	CTS	从调制解调器到终端	辅信道清除发送
14	TXD	从终端到调制解调器	辅信道发送数据
15		从调制解调器到终端	发送器信号定时
16	RXD	从调制解调器到终端	辅信道接收数据
17		从调制解调器到终端	接收器信号定时
18	—	———	未定义
19	RTS	从终端到调制解调器	辅信道请求发送
20	DTR	从终端到调制解调器	数据终端就绪
21		从调制解调器到终端	信号质量检测
22		从调制解调器到终端	振铃指示
23		从终端到调制解调器 从调制解调器到终端	数据信号速率选择器
24		从终端到调制解调器	发送器信号定时
25	—	———	未定义

2、电气特性

RS－232C 采用负逻辑，即：

逻辑“1”： －5V～－15V

逻辑“0”： ＋5V～＋15V

表 7-2 列出了 RS－232C 接口的主要电气性能。

表 7-2 RS－232C 电气特性表

带 3-7kΩ负载时驱动器的输出电平	逻辑 1:－5V 到－15V 逻辑 0:＋5V 到＋15V
不带负载时驱动器的输出电平	－25V 到＋25V
驱动器通断时的输出阻抗	>300Ω
输出短路电流	<0.5A
驱动器转换速率	<30V/us
接收器输入阻抗	在 3kΩ到 7kΩ之间
接收器输入电压的允许范围	－25V 到＋25V
输入开路时接收器的输出	逻辑 1
输入经 330Ω接地时接收器的输出	逻辑 1
＋3V 输入时接收器的输出	逻辑 0
—3V 输入时接收器的输出	逻辑 1
最大负载电容	2500pF

3、 用 RS－232C 总线连接系统

用 RS－232C 总线连接系统时，有近程通信方式和远程通信方式。近程通信是指传输距离小于 15 米的通信，这时可以用 RS－232C 电缆直接连接。15 米以上的长距离通信，需要采用调制解调器（MODEM）。

图 7-6 为最常见的采用调制解调器的远程通信连接。

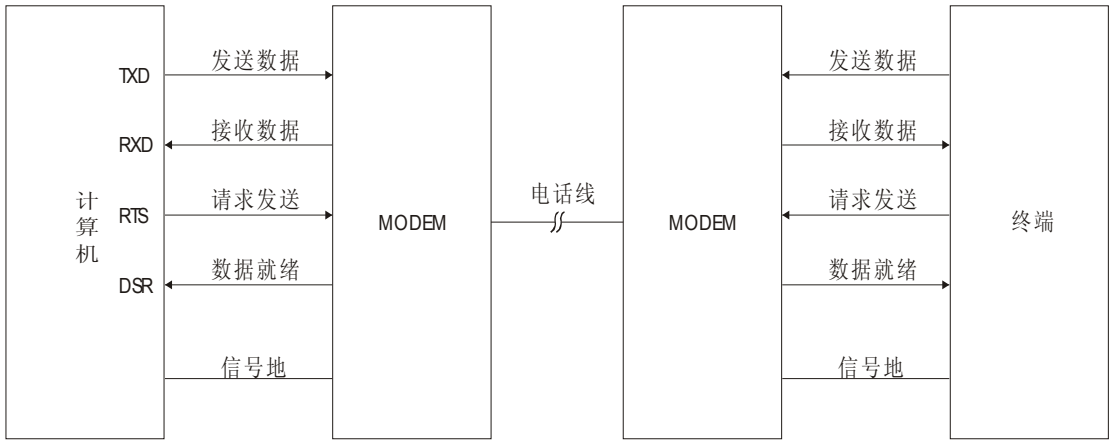


图 7－6 计算机与终端的远程连接

当计算机与中断之间利用 RS－232C 作近程连接时，这时有几根线实现交换连接。

图 7-7 为计算机与终端之间利用 RS－232C 连接的最常用的交叉连线图。图中“发送数据”与 “接收数据”是交叉相连的， 使得两台设备都能正确的发送和接收。“数据终端就绪”与“数据装置就绪”两根线也是交叉相连的， 使得两设备都能检测出对方是否已经准备好。

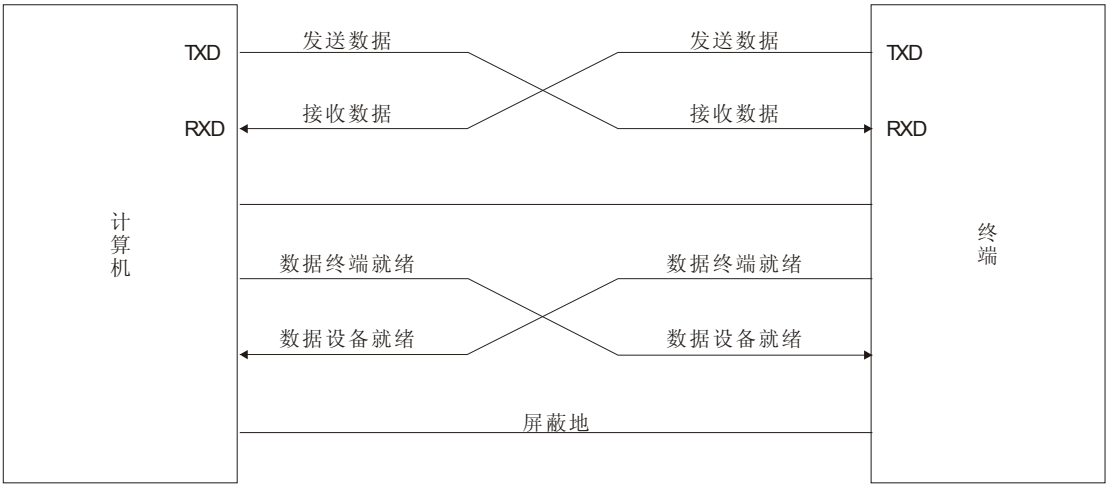


图 7－7 计算机与终端的 RS-232C 连接

图 7-8 是比图 7-7 更完整的一个连接图。由图可见，它们的“请求发送”端与自己的“清除发送”端相连，使得当设备向对方请求发送时，随即通知自己的“清除发送”端，表示对方已经响应。这里的请求发送线还连往对方的“载波检测”线，这是因为“请求发送”信号的出现类似于通信通道中的载波检出。图中的“数据设备（装置）就绪”是一个接收端，它与对方的“数据终端就绪”相连，就能得知对方是否已经准备好。“数据设备就绪”端收到对方“准备好”的信号，类似于通信中收到对方发出的“响铃指示”的情况，因此可将“响铃指示”与“数据设备就绪”并联在一起。

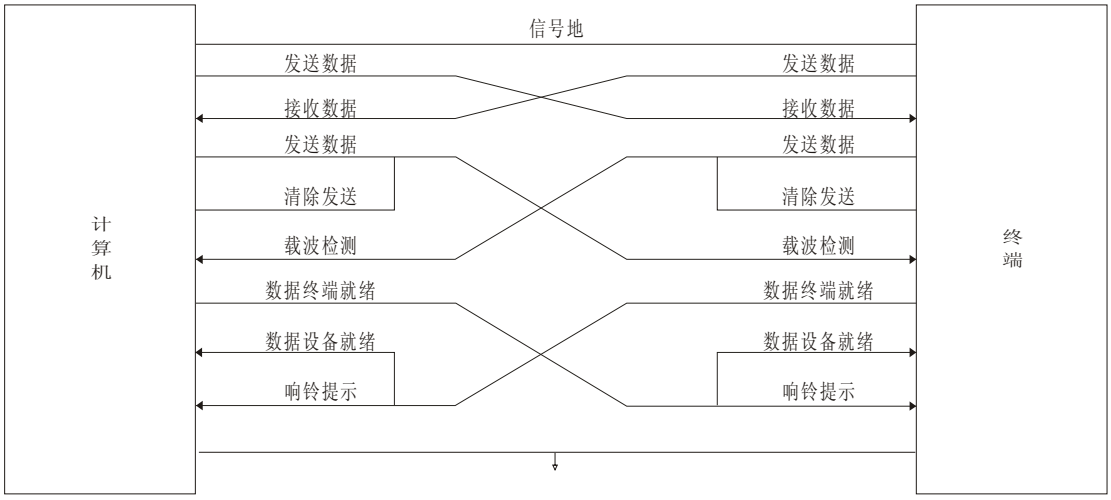
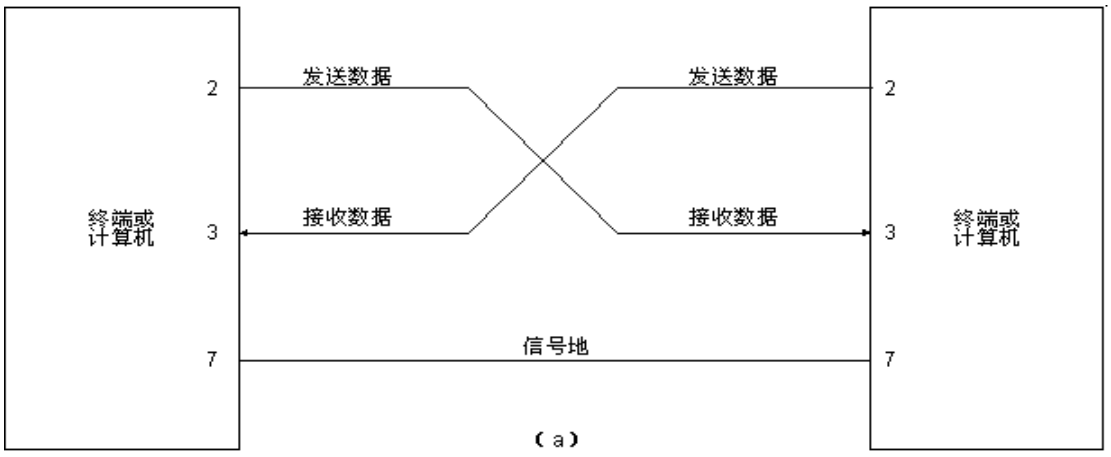


图 7－8　终端/计算机与终端/计算机的更完整接连

还有最简单的应用接法。如图 7-9 所示，仅将“发送数据”与“接收数据”交叉连接，其余的信号均不用。图 7-9a 为其余信号都不连接的形式，图 7-9b 中，同一设备的“请求发送”被连到自己的“清除发送”及“载波检测”，而它的“数据终端就绪”连到自己的“数据设备就绪”。

图 7-9a 的连接方式不适用于需要检测“清除发送”、“载波检测”、“数据设备就绪”等信号状态的通信程序；对图 7-9b 的连接，程序虽可运行下去，但并不能真正检测到对方状态，只是程序受到该连接方式的欺骗而已。在许多场合下只需单向传送时，例如计算机向单片机开发系统传送目标程序，就采用图 7-9 的连接方式进行通信。



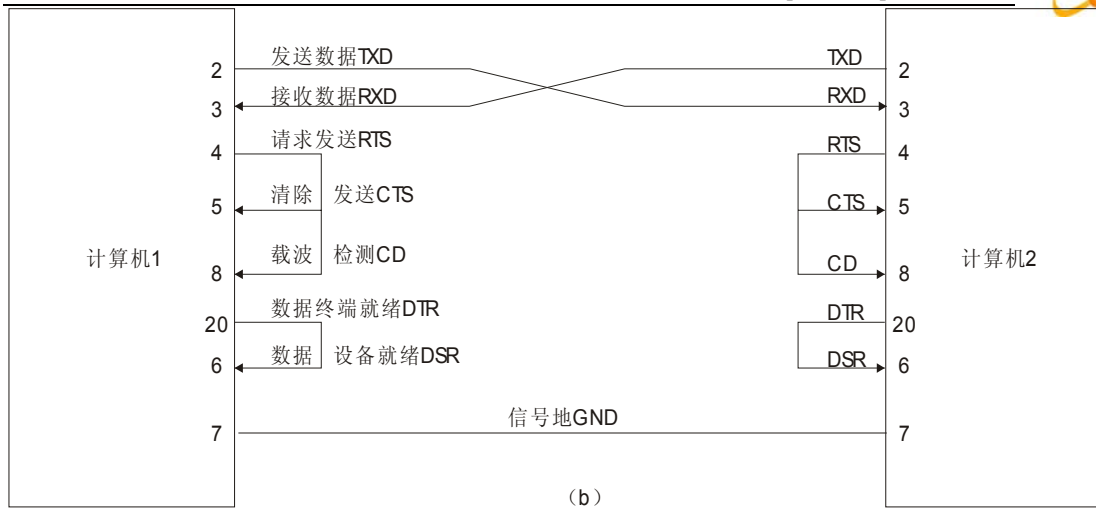


图 7－9　终端/计算机与终端/计算机连接的最简单形式

4、 RS－232C 标准接口的实现及电平转换

构成 RS－232C 标准接口的硬件有多种，如 INS8250、Intel8251A、Z80-SIO 等通信接口芯片。通过编程，可使其满足 RS－232C 通信接口的要求，TL16C552 是德洲仪器公司新推出的一款串行异步通信控制芯片，其功能强大，我们将在后面给出应用的例子进行详细的分析。

由于 232 使用的是负电平，不能满足 TTL 电平的传送要求，因此，在与单片机通信时还需要一块电平转换芯片，常用的这种芯片有 MC1488,MC1489 和 MAX232 等。

7.3　单片机串行接口

51 单片机除了具有 4 个 8 位并行口外，还具有一个功能强大的全双工串行接口，此串口能同时进行串行发送和接收数据。它既可以做 UATR（通用异步接收和发送器）用，也可以做同步移位寄存器用。使用串行口可以实现 8051 单片机系统之间点对点的单片机通信和 8051 与系统机的单机或多机通信。该串口有 4 种工作方式，以供不同场合使用。波特率可由软件设置，由片内的定时器/计数器产生。接收、发送均可工作在查询方式或中断方式，使用十分灵活。

7.3.1　串行口结构

8051 通过引脚 RXD（P3.0，串行数据接收端）和引脚 TXD（P3.1，串行数据发送端）与

外界进行通信。其内部结构简化示意图如图 7-11 所示。图中有两个物理上独立的接收、发送缓冲器 SBUF，它们占用同一地址 99H，可同时发送和接收数据。发送缓冲器只能写入，不能读出；接收缓冲器只能读出，不能写入。

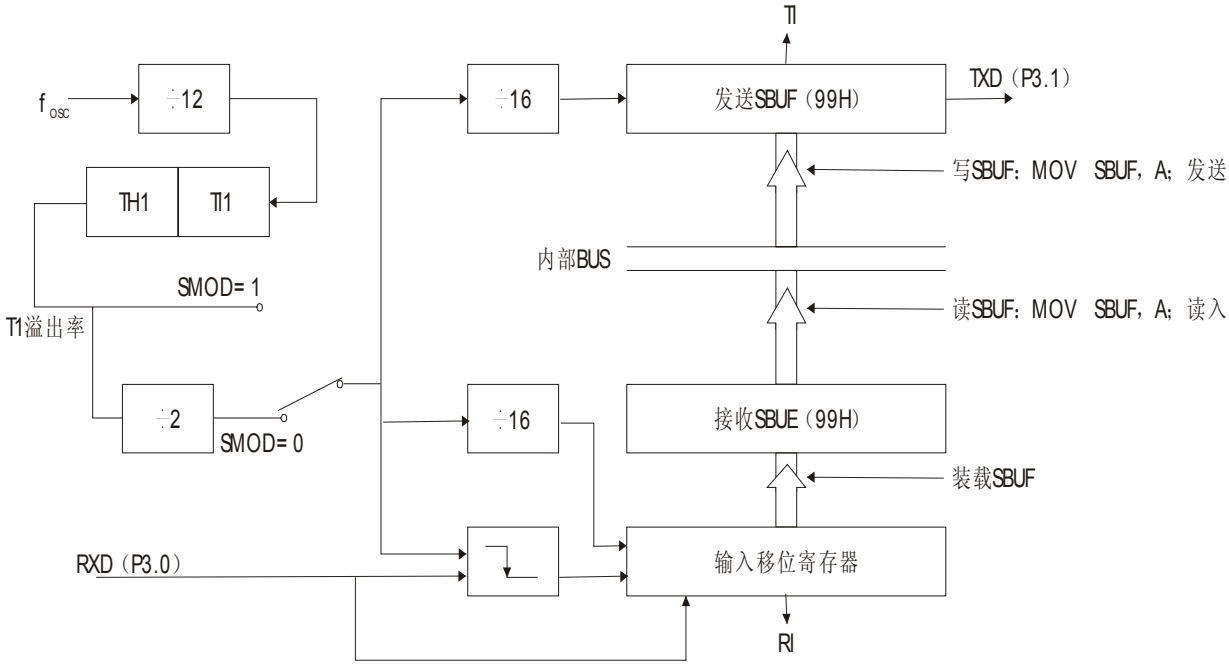


图 7-11 串行口内部结构示意图

串行发送和接收的速率与移位时钟同步。8051 用定时器 T1 作为串行通信的波特率发生器，T1 溢出率经 2 分频（或不分频）后又经 16 分频作为串行发送或接收的移位脉冲。移位脉冲的速率即波特率。

从图中可以看出，接收器是双缓冲结构，在前一个字节被从接收缓冲器 SBUF 读出之前，第二个字节开始被接收（串行输入到移位寄存器），但是，在第二个字节接收完毕而前一个字节 CPU 未读取，会丢失前一字节。

串行口的发送和接收都是以特殊功能寄存器 SBUF 的名义进行读或 写的。当向 SBUF 发“写”命令时（执行“MOV A，SBUF”指令），即是向发送缓冲器 SBUF 装载并开始由 TXD 引脚向外发送一帧数据，发送完便使发送中断标志位 TI＝1。

在满足串行口接收中断标志位 RI（SCON.0）＝0 的条件下，置允许接收位 REN（SCON.4）＝1 就会接收一帧数据进入移位寄存器，并装载到接收 SBUF 中，同时使 RI＝1。当发读 SBUF 命令时（执行“MOV A，SBUF”指令），便由接收缓冲器（SBUF）取出信息通过 8051 内部总线送 CPU。

对于发送缓冲器，因为发送时 CPU 是主动的，不会产生重叠错误，一般不需要用双缓冲结构来保持最大传送速率。

7.3.2 串行口控制寄存器

串行口有两个特殊功能寄存器 SCON 和 PCON

1、SCON

该寄存器的字节地址为 98H，可位寻址。SCON 格式为

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SCON	SM0	SM1	SM2	REN	TB8	RB8	TI	RI
位地址	9F	9E	9D	9C	9B	9A	99	9B

SM0、SM1：控制串行口的工作方式。其功能详见下表：

表 7-3

SM0	SM1	方 式	功能说明
0	0	0	移位寄存器方式（用于扩展I/O口）
0	1	1	8位UART，波特率可变（T1溢出率 / n）
1	0	2	9位UART，波特率为$f_{osc}/64$或$f_{osc}/32$
1	1	3	9位UART，波特率可变（T1溢出率 / n）

SM2：允许方式 2 和方式 3 进行多机通信控制位。在方式 2 和方式 3 中，如 SM2＝1,则接收到的第 9 位数据（RB8）为 0 时不激活 RI。在方式 1 时，如 SM2=1,则只有收到有效停止位时才会激活 RI。若没有接收到有效停止位，则 RI 清 0。方式 0 中，SM 应为 0。

REN：允许串行接收控制位。由软件置位时允许接收，由软件清 0 时禁止接收。

TB8：是工作在方式 2 和方式 3 时要发送的第九位数据，根据需要由软件置位和复位。

RB8：是工作在方式 2 和方式 3 时接收到的第 9 位数据，在方式 1，如果 SM2=0，RB8 是接收到的停止位，在方式 0 不使用 RB8。

TI：发送中断标志位。由片内硬件在方式 0 串行发送第 8 位结束时置位，或在其它方式串行发送停止位的开始时置位。必须由软件清 0。

RI：接收中断标志位。由片内硬件在方式 0 串行接收第 8 位结束时置位，或在其它方式串行接收停止位的中间时置位。必须由软件清 0。

SCON 复位值为 00H。

图 7-12 是 SCON 的功能说明。

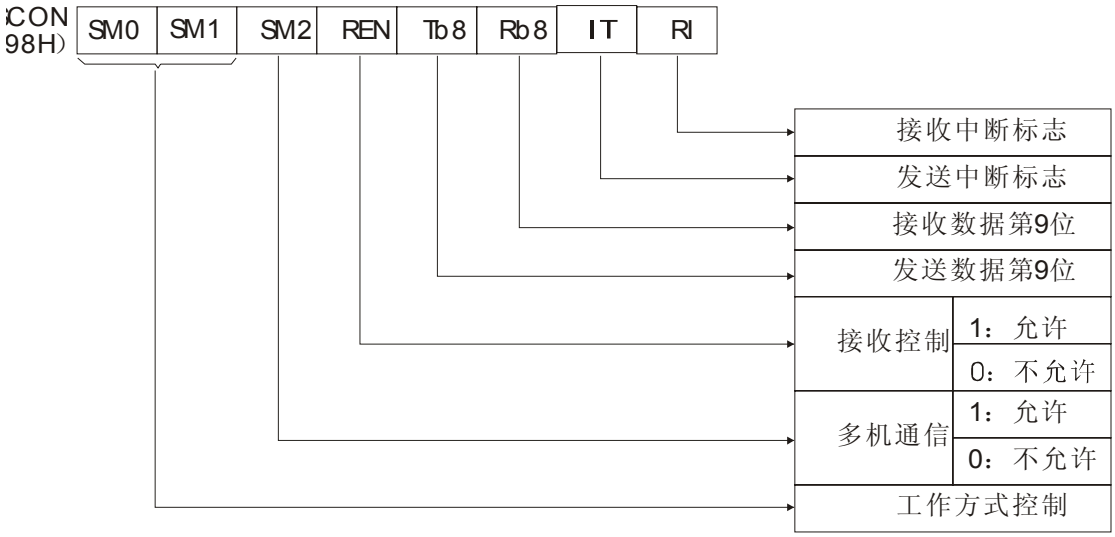


图 7－12　串行口控制寄存器 SCON

2、 PCON

字节地址 87H，没有位寻址功能。PCON 的格式如下，其中与串行接口有关的只有 D7 位：

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
PCON	SMOD							

SMOD：波特率选择位。在串行口方式 1、方式 2、和方式 3 时，波特率和 2^{SMOD} 成正比，也就是当 SMOD＝1 时，波特率提高一倍。SMOD 复位值为 0。

7.3.3　波特率设置

51 单片机的波特率设置较为灵活，它除了与系统的振荡频率 f_{OSC}，电源控制寄存器 PCON 的 SMOD 有关外，还与定时器 T1 的设置有关。

在串行口工作在方式 0 时，波特率不需设定，只与系统振荡频率有关，其大小为 $f_{OSC}/12$。

在方式 2 ，波特率也固定为两种：

$$当SMOD=1时，波特率=\frac{2^{SMOD}}{64}\ f_{OSC}=\frac{f_{OSC}}{32}$$

$$当SMOD=0时，波特率=\frac{2^{SMOD}}{64}\ f_{OSC}=\frac{f_{OSC}}{64}$$

只有在方式 1 和方式 3，波特率才是可以改变的：

$$波特率=\frac{2^{SMOD}}{32}\times 定时器T1的溢出率$$

为了灵活的设置波特率，我们一般使串口工作在方式 1 和方式 3，为确定波特率，我们必须计算出 T1 的溢出率。

T1 溢出率的计算

溢出率是指每秒定时器溢出的次数，定时器 T1 的溢出率与它的操作模式有关。定时器 T1 作波特率发生器时，通常选用定时器模式 2（自动重装初值定时器）比较实用。在模式 2 时，定时器的溢出率由下式计算得出。

$$定时器\ T1\ 的溢出率 = f_{OSC}/12 /(2^8-X)$$

X 为定时器 T1 的计数初值。

波特率的设置

当串行口工作在方式 1 和方式 3 时，波特率应由下式决定：

$$波特率=\frac{2^{SMOD}}{32}\times 定时器T1的溢出率=\frac{2^{SMOD}}{32}\times\frac{f_{OSC}}{12\times(2^8-N)}$$

在实际应用中，往往是给定通信波特率，而后去定定时器初值，由上式可得：

$$N=256-\frac{2^{SMOD}\times f_{OSC}}{波特率\times 32\times 12}$$

N 为时间常数

例如，若系统时钟频率 f_{OSC} 为 6MHz，当 SMOD＝1，通信波特率为 2400 时，时间常数

$$N=256-\frac{2^1\times 6\times 10^6}{2400\times 32\times 12}=242.98\sim 243=F3H$$

初始化程序如下：

```
INIT：    MOV    TMOD,#20H      ；选择定时器 T1 模式 2,计时方式
          MOV    TH1,#0F3H      ；预置时间常数
          MOV    TL10F3H
          SETB    TR1           ；启动定时器 T1
        MOV    PCON,#80H        ；SMOD＝1
        MOV    SCON,#50H        ；串行口方式 1 工作
```

执行上面的程序后，即可完成对定时器 T1 的操作模式及串行口工作方式和波特率的设置。

7.3.4 串行口的工作方式

串行口的工作方式有四种，由 SCON 中的 SM0、SM1 定义，编码及功能前面已经说明。在这四种工作方式中，串行通信只使用方式 1、2、3。 方式 0 主要用于扩展并行输入输出口。

1、 工作方式 0

方式 0 为同步移位寄存器输入/输出方式，常用于扩展 I/O 口。串行数据通过 RXD 输入或输出，而 TXD 用于输出移位时钟，作为外接部件的同步信号。图 7-13（a）为发送电路，图 7-14(a)为接收电路。这种方式不适用于两个 8051 之间的直接数据通信，但可以通过外接移位寄存器来实现单片机的接口扩展。例如，74LS164 可用于扩展并行输出口，74LS165 可用于扩展输入口。在这种方式下，收/发的数据为 8 位，低位在前，无起始位、奇偶校验位及停止位，波特率是固定的。

发送过程中，当执行一条将数据写入发送缓冲器 SBUF（99H）的指令时，串行口把 SBUF 中的 8 位数据以 fosc/12 的波特率从 RXD（P3.0）端输出，发送完毕置中断标志 TI＝1。方式 0 发送时序如图 7-13(b)所示。写 SBUF 指令在 S6P1 处产生一个正脉冲，在下一个机器周期的 S6P2 处数据的最低位输出到 RXD（P3.0）脚上；再在下一个机器周期的 S3，S4，S5 输出移位时钟为低电平，而在 S6 及下一个机器周期的 S1，S2 为高电平，就这样将 8 位数据由低位到高位一位一位顺序通过 RXD 线输出，并在 TXD 脚上输出 fosc/12 的移位时钟。在“写 SBUF”有效后的第 10 个机器周期的 S1P1 将发送中断标志 TI 置位。图中，74LS164 是 TTL “串入并出”移位寄存器。

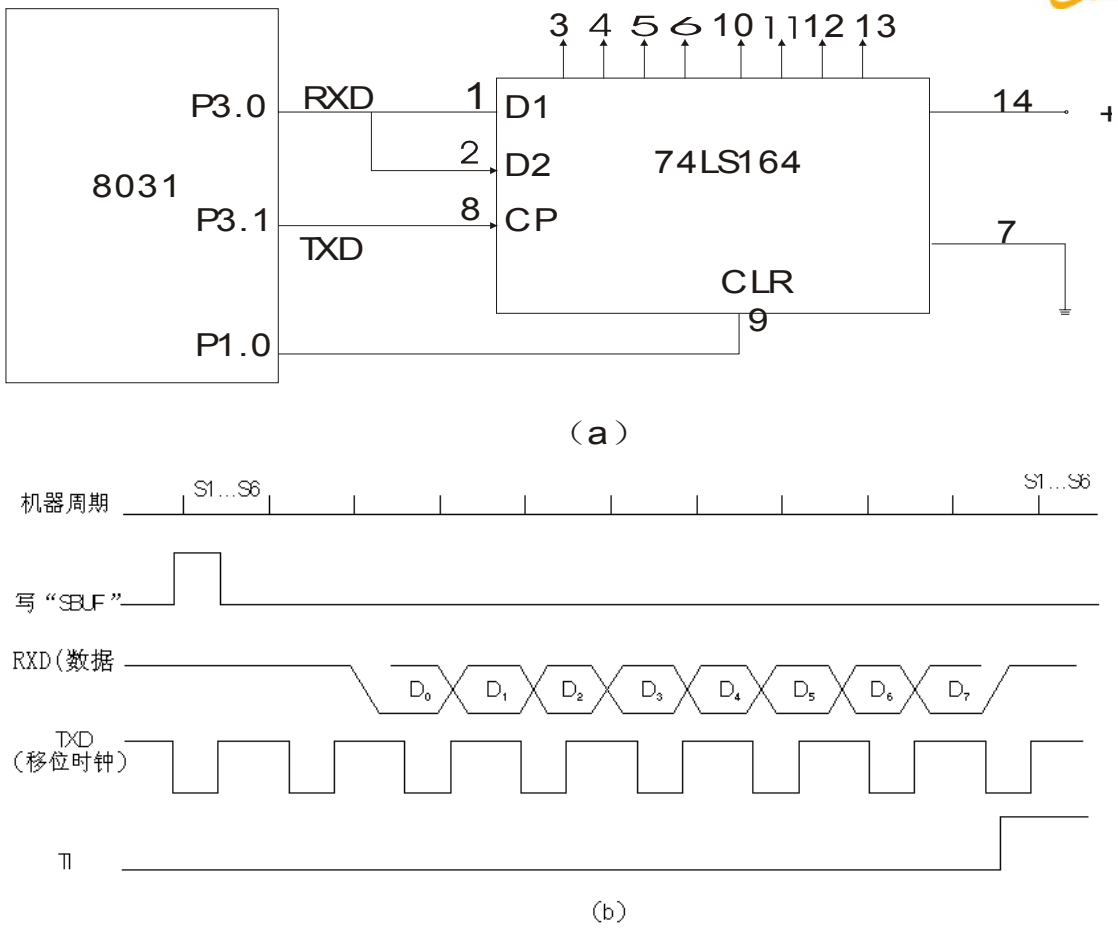
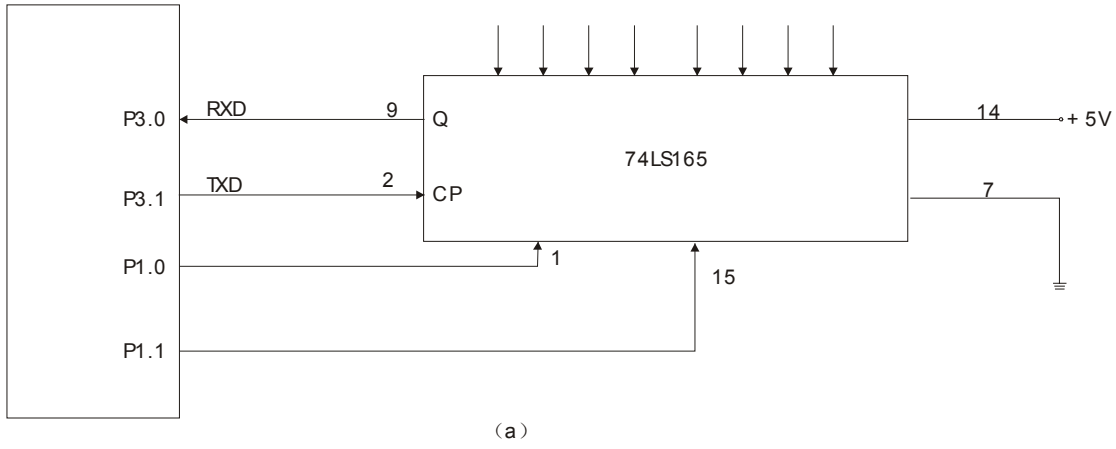


图 7-13　　方式 0 发送电路时序



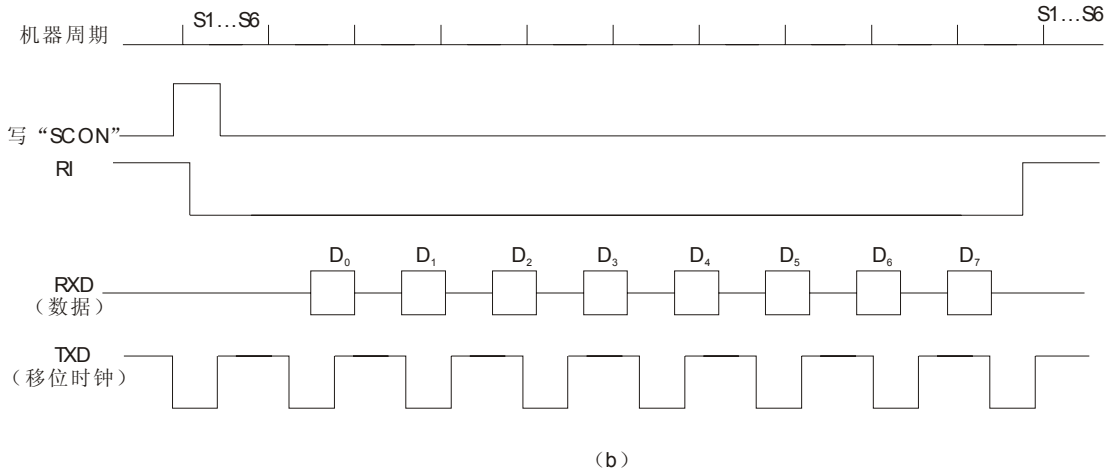


图 7－14　方式 0 接收电路及时序

2、 工作方式 1

当 SCON 中的 SM0、SM1 两位为 01 时，串行口以方式 1 工作，此时串行口为 8 位异步通信接口。一帧信息为 10 位：1 位起始位，8 位数据位（低位在先）和一位停止位。TXD 为发送端，RXD 为接收端，波特率可变。

1. 方式 1 发送

串行口以方式 1 发送时，数据由 TXD 端输出，CPU 执行一条写入 SBUF 的指令就启动串行口发送，发送完一帧信息时，发送中断标志置 1,其时序如图 7-15 所示。

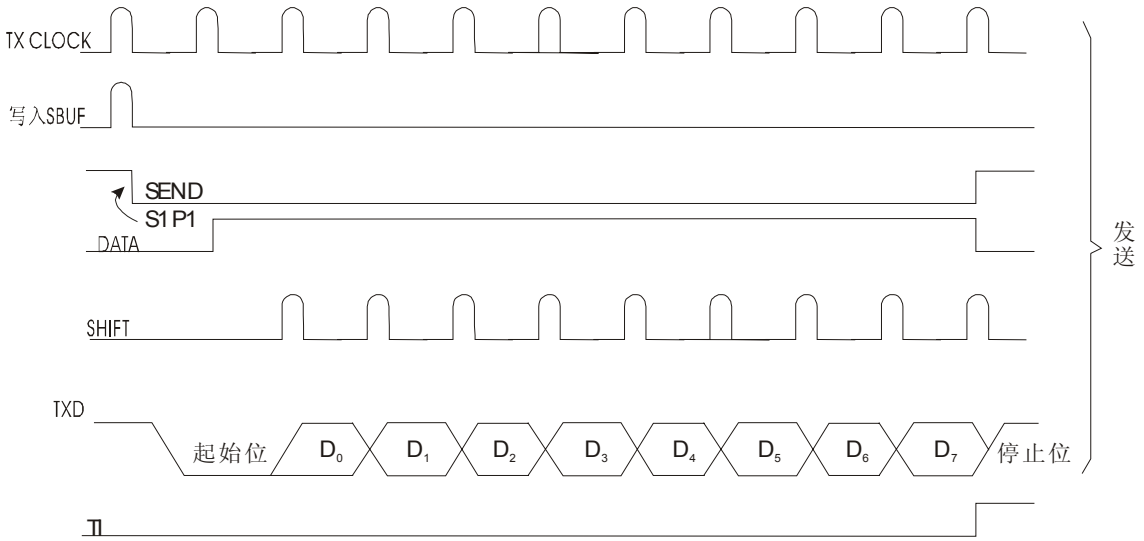


图 7－15　方式 1 发送时序

2. 方式 1 接收

当 REN＝1 时，接收器便以所选波特率的 16 倍速率采样 RXD 引脚状态，当采样到 RXD 从 1 到 0 的跳变时就启动接收器接收，接收的值是 3 次采样中至少 2 次相同的值，以保证可靠无误。在起始位，如果接收到的值不为 0,则起始位无效，复位接收电路，当再次接收到一个由 1 到 0 的跳变时，重新启动接收器。如果接收值为 0,起始位有效，接收器开始接收本帧的其余信息（一帧信息为 10 位）。在方式 1 接收中，若同时满足以下两个条件：

RI＝0;

SM2=0 或接收到的停止位为 1。

则接收数据有效，实现装载 SBUF、停止位进入 RB8、RI 置 1。接收控制器再次采样 RXD 的负跳变，以便接收下一帧数据。如果上述两个条件任一不满足，信息将丢失。中断标志 RI 必须由用户的软件清 0,通常情况下，串行口以方式 1 工作时，SM2 置位 0,方式 1 的接收时序见图 7-16。

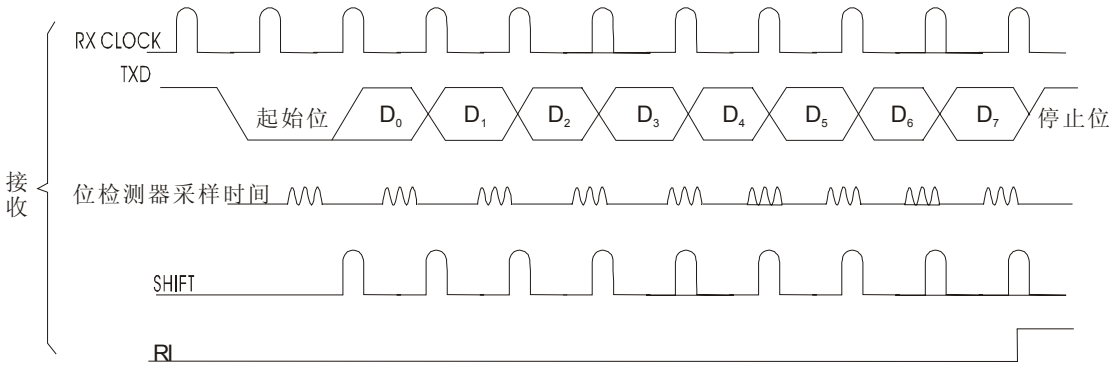


图 7－16　方式 1 接收时序

3. 工作方式 2 和方式 3

串行口工作在方式 2 和方式 3 均为每帧 11 位异步通信格式，由 TXD 和 RXD 发送与接收（两种方式操作是完全一样的，所不同的只是波特率）。每帧 11 位，即 1 位起始位，8 位数据位（低位在前），1 位可编程的第 9 数据位和 1 位停止位。发送时，第 9 数据位（TB8）可以设置为 1 和 0,也可以将奇偶位装入 TB8，从而进行奇偶校验；接收时，第 9 数据位进入 SCON 的 RB8。

方式 2 和方式 3 的发送、接收时序如图 7-17 所示。其操作与方式 1 类似。

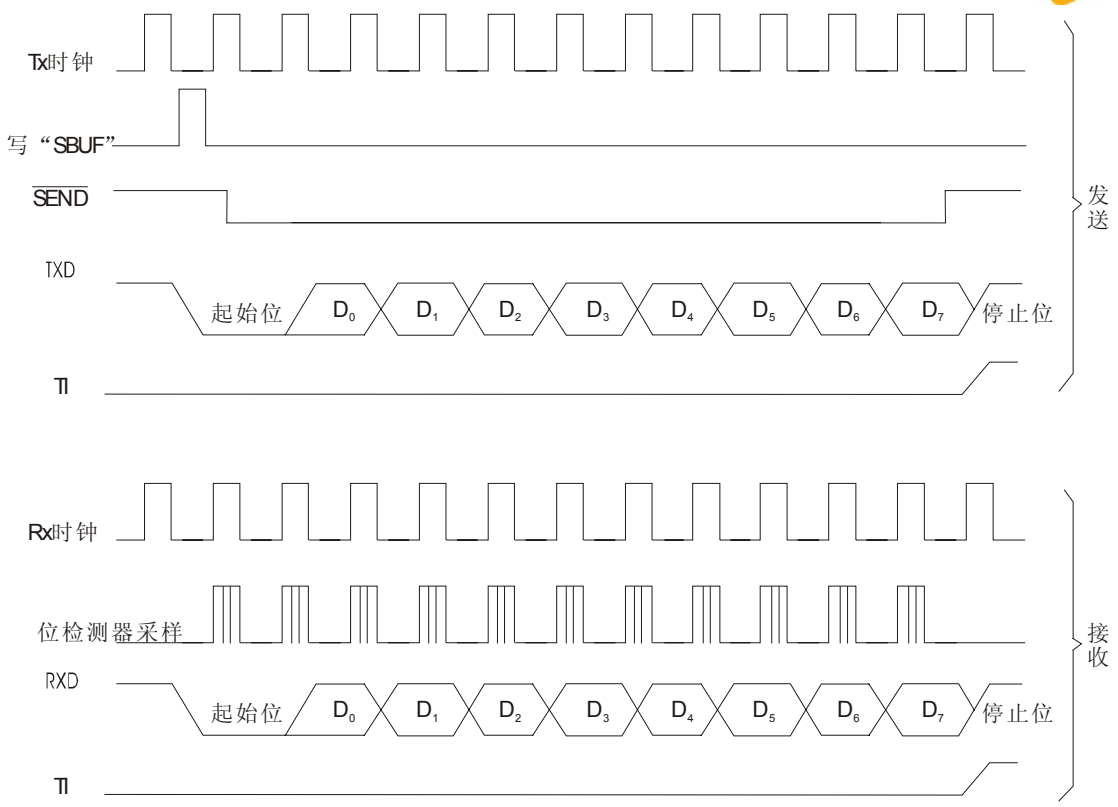


图 7－17　方式 2、方式 3 发送和接收时序

发送前，先根据通信协议由软件设置 TB8（如作奇偶校验位或地址/数据标志位），然后将要发送的数据写入 SBUF，即可启动发送过程。串行口能自动把 TB8 取出，并装入到第 9 位数据位的位置，再逐一发送出去。发送完毕，使 TI＝1。

接收时，使 SCON 中的 REN＝1,允许接收。当检测到 RXD（P3.0）端有 1——0 的跳变（起始位）时，开始接收 9 位数据，送入移位寄存器（9 位）。当满足 RI＝0 且 SM2＝0,或接收到的第 9 位数据为 1 时，前 8 位数据送入 SBUF，附加的第 9 位数据送入 SCON 中的 RB8,置 RI 为 1；否则，这次接收无效，也不置位 RI。

7.4　单片机与 PC 机的通信

在由 PC 机与单片机组成的系统中，常要涉及通信问题，在近距离通信时常采用 232 通讯方式，而远距离且有强干扰时常采用 485 通讯。

7.4.1　异步通讯适配器

由于 PC 机没有象单片机那样提供了串口,所以在与单片机串行通信的时候需要一片异步通讯适配器来控制串行数据的传送格式和速度。常见的异步通讯芯片有 8250，最近德州仪器

公司新推出一款通用异步通讯芯片 TL16C552，这款芯片具有与 8250 相似的操作却有比 8250 更强的功能。它能提供两个串口和一个并口，现将它的操作及控制介绍如下。

TL16C552 是美国 TI 公司生产的异步通信控制芯片，它有如下功能特点：

- 增强的双向打印机端口
- 16 字节的 FIFO 可减少 CPU 中断
- 每个通道具有独立的发送、接收、线路状态和设置中断功能
- 每个通道具有独立的 MODEM 控制信号
- 每个通道具有可编程串行数据发送格式
  ——数据位长度为 5/6/7/8
  ——偶校验、奇校验或无校验
  ——停止位长度为 1/1.5/2
- 可编程波特率发生器
- 每个通道对数据和控制总线具有三态 TTL 驱动

TL16C552 的管脚及功能

TL16C552 是 68 针 PLCC(Plastic Leaded Chip Carrier)封装，其管脚分布如图 7-18 所示。从图中我们可以看出，它的串口主要完成两项功能即把从外部设备接收进来的串行数据转换成并行数据；以及把 CPU 的并行数据转换成串行数据以利发送。在正常操作的过程中，CPU 每时每刻都可以读 16C552 的完成状态。状态信息报告 TL16C552 传输操作的类型和状态，还包括各种错误状态，如奇偶校验、溢出、格式错误或停顿指示等。

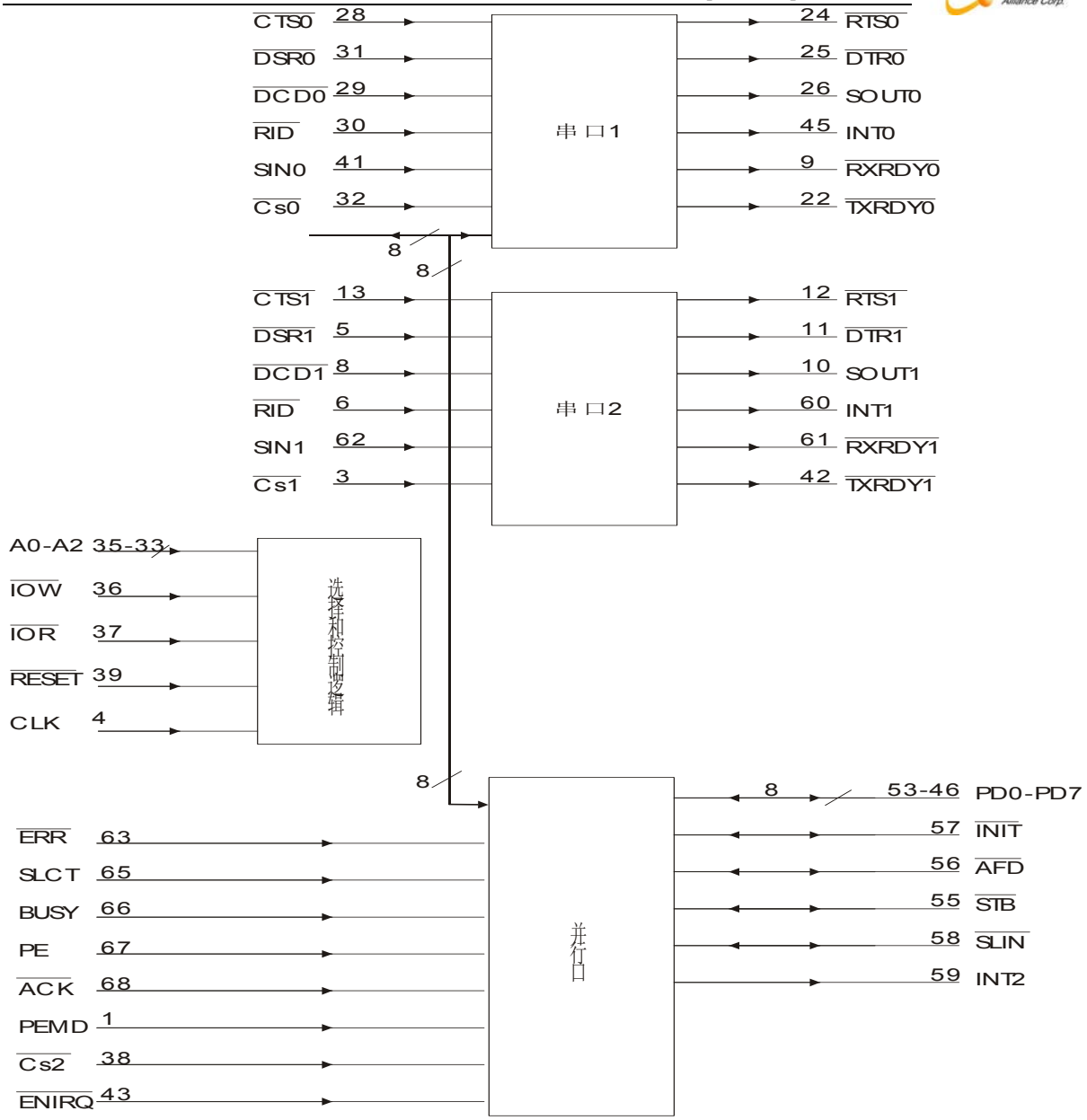


图 7－19　功能框图

TL16C552 内部对串行口操作有 11 个单字节寄存器。CPU 执行输入输出可以访问这些寄存器。串行口的这 11 个寄存器占用了 7 个 I/O 口地址。有些寄存器共同使用一个端口地址，可以通过读写信号和线路控制寄存器的 D7（DLAB）来区分。具体描叙见下表：

表　串行接口寄存器

DLAB	A2	A1	A0	符 号	寄 存 器
L	L	L	L	RBR	接收缓冲寄存器
L	L	L	L	THR	发送保存寄存器
L	L	L	H	IER	中断允许寄存器
X	L	H	L	IIR	中断识别寄存器
X	L	H	L	FCR	FIFO 控制寄存器
X	L	H	H	LCR	线路控制寄存器
X	H	L	L	MCR	MODEM 状态寄存器
X	H	L	H	LSR	线路控制寄存器
X	H	H	L	MSR	MODEM 状态寄存器
H	L	L	L	DLL	除数锁存寄存器低位
H	L	L	H	DLM	除数锁存寄存器高位

注：1.当 TL16C552 的 CS0 或 CS1 是低电平时，串行接口才被选中

2.X＝无关 L＝低电平 H＝高电平

下面对每个寄存器作一简单介绍。

接收缓冲寄存器存放接收到的并且已经转换过的并行数据。线路状态寄存器的 D0 指明该寄存器是否已经接收到一个完整的字符。

发送保持寄存器存放将要发送的数据，和发送缓冲寄存器有关的标志是线路状态寄存器的 D5。当 D5＝1,说明发送缓冲寄存器空，可输入下一个字符。

有四种类型的事件能够引发串行接口中断请求，设置中断允许寄存器可以禁止某些中断源提出中断请求。在允许中断地条件下，如果有多个中断源申请中断，中断识别寄存器能够区分这些中断类型。

设置中断允许寄存器前必须将线路控制寄存器的 D7 清为 0。中断允许寄存器仅使用低 4 位。

D0＝1 允许接收就绪中断

D1=1 允许发送缓冲器空中断

D2＝1 允许接收出现错误或接收到间断信号中断

D3＝1 允许 MODEM 状态中断

四种类型的中断对应 4 级优先权，在中断识别寄存器中高 5 位不用，恒为 0,D0＝0 表示有中断产生，D0＝1 表示无中断产生，D0 与其它位的组合定义如下表：

IIR 的功能

D3D2D1D0	优先级	中断类型	复位

0	0	0	1	无	无	无
0	1	1	0	1	接收状态错	读 LSR
0	1	0	0	2	接收数据就绪	读 RBR
1	1	0	0	2	字符空指示	读 RBR
0	0	1	0	3	发送完毕	写 THR
0	0	0	0	4	MODEM 状态变化	读 MSR

FIFO 控制寄存器是一个只写寄存器，它允许和清除 FIFO，设置接收器 FIFO 触发标准和选择 DMA 信号的类型，对该寄存器的各位描叙如下：

D0 允许发送器和接收器的 FIFO，该位为 0，在两者 FIFO 中的所有数据都被清除，对该寄存器的其它位编程时，必须将 D0 设置为 1。

D1＝1 清除在接收器 FIFO 中的所有字节和重新设置计数器。

D2＝1 清除在接收器 FIFO 中的所有字节和重新设置计数器。

D3＝1 当 D0＝1 时，将 RXDRY 和 TXDRY 从模式 0 改为模式 1。

D4、D5 保留

D6、D7 设置接收器 FIFO 中断地触发标准。

线路控制寄存器存放传送的二进制位串数据格式，各位定义如图 7-19 所示。

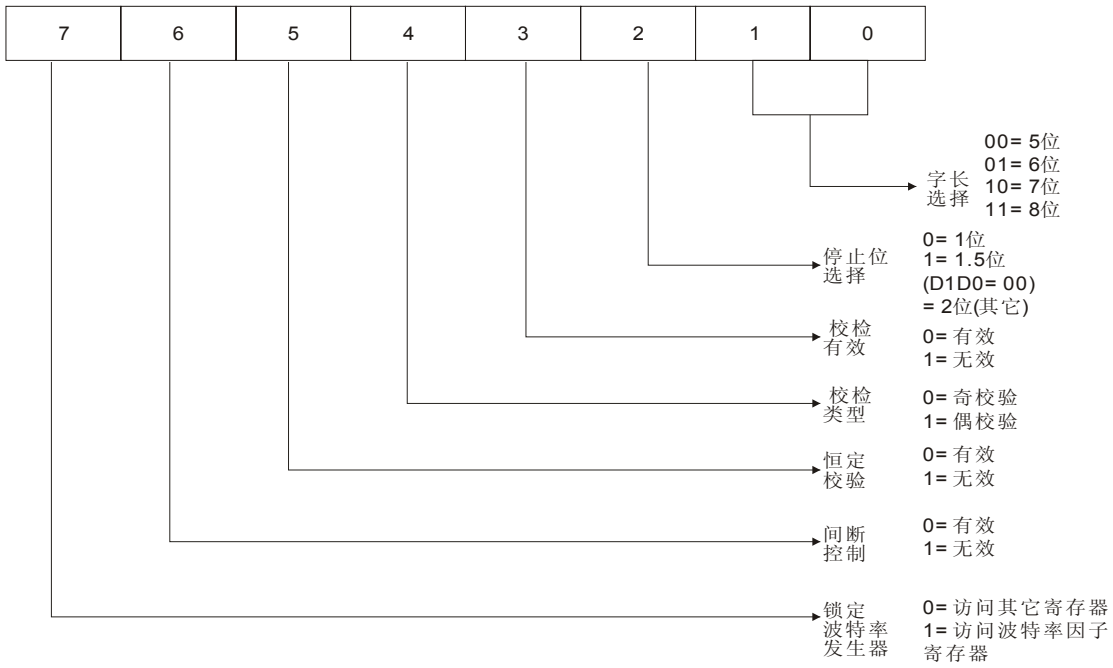


图 7-20　LCR 各位定义

线路状态寄存器记录有串行数据发送和接收过程的状态信息，CPU 可以在任何时候读取这些信息。该寄存器的状态位如下表所示：

LSR 的各位定义

LSR 的各位	1	0
D0 接收完成标志	接收完毕	读后复位
D1 接收重叠标志	发生重叠	读后复位
D2 奇偶校验错标志	奇偶校验	读后复位
D3 格式错标志	接收有错	接收无错
D4 间断标志	连续接收到 0	未间断
D5 发缓冲器空标志	THR 已空	写 THR 复位
D6 发移位寄存器空标志	已空	未空
D7 接收器 FIFO 错	FIFO 有错	FIFO 无错

MODEM 控制寄存器主要用来控制从 TL16C552 输出引脚输出电平，D3～D0 分别对应 TL16c552 的 4 个输出引脚。

D0＝1　DTR 引脚输出低电平
D1＝1　RTS 引脚输出低电平
D2＝1　OUT1 引脚输出低电平
D3＝1　OUT2 引脚输出低电平
D4＝1　自测循环回送状态
D4＝0　正常工作
该寄存器可用于 TL16C552 串口的测试。

MODEM 状态寄存器记录着 TL16C552 的当前状态和变化状态信息。变化状态是指 CPU 读取 MODEM 状态寄存器后，这 4 位输入引脚的电平发生了变化。该寄存器用低 4 位记录并保持这种变化。

D0＝1　CTS 电平发生变化
D1＝1　DSR 电平发生变化
D2＝1　RI 电平发生变化
D3＝1　DCD 电平发生变化
D4＝1　CTS 为低电平
D5＝1　DSR 为低电平
D6＝1　RI 为低电平
D7＝1　DCD 为低电平

两个波特率因子寄存器构成一个 16 位的波特率因子寄存器。在 TL16C552 内部具有波特率发生器，产生发送数据的时钟信号，波特率因子可以通过下列算式求出：

波特率因子＝基准时钟频率/（16×波特率）

这个波特率发生器可以利用比较通用的三种不同频率产生标准的波特率。这三种不同频

率是 1.8432MHz、3.072MHz 和 8MHz。另外可以任意选择写入波特率因子高字节或低字节的顺序，但写入前必须设置线路控制寄存器的 D7＝1.写入波特率因子后，应将线路控制寄存器的 D7 恢复为 0,以便于访问其它寄存器。

7.4.2 电平转换芯片

由于 232 通信使用的是负电平，所以单片机与 PC 机通信时中间必须有电平转换芯片进行电平转换。能进行电平转换的芯片有 MC1488,75188，实现 TTL－232C 的电平转换，MC1489,75189 实现 232C－TTL 的电平转换。由于这两款芯片在使用时需接正负 15V 或正负 12V 电压，不是十分方便，它的应用不如只需单电源供电的电平转换芯片方便。使用单电源的电平转换芯片有 SIPEX 公司的 SP232、INTERSIL 公司的 ICL232、和 MAXIM 公司的 MAX232 等。

下面我们给出 SIPEX 公司的 SP232E 的部分特点并附一份实际电路原理图。

特点：

单电源＋5V 电压供电

符合所有 RS-232D 和 ITU V.28 特性

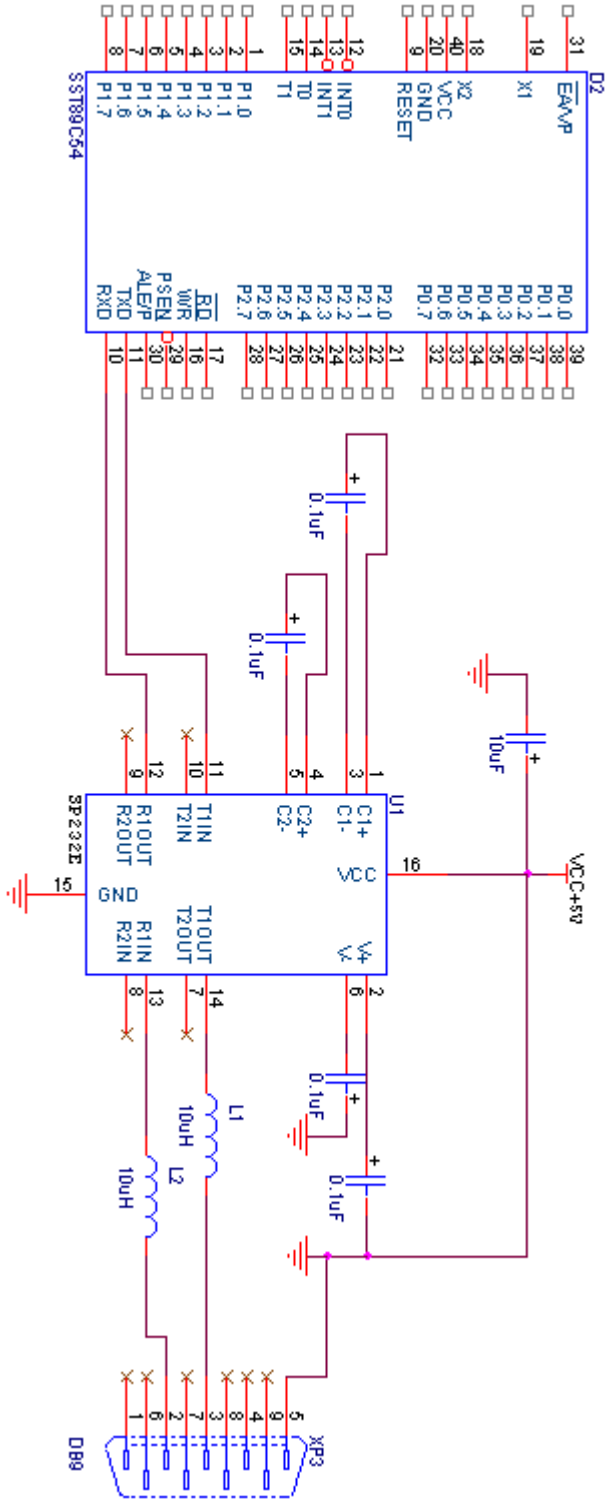
操作电容 0.1uF 到 10uF

高数据率达 120KBPS

三态、TTL/CMOS 接收和输出

低功耗 3mA 电流供电

改进的 ESD 特性


D2
SST89C54
P1.0
P1.1
P1.2
P1.3
P1.4
P1.5
P1.6
P1.7
INT0
INT1
T0
T1
RESET
VCC
GND
X2
X1
EA\VP
P0.0
P0.1
P0.2
P0.3
P0.4
P0.5
P0.6
P0.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
RD
WR
PSEN
ALE/P
TXD
RXD
0.1uF
0.1uF
10uF
U1
SP232E
C1+
C1-
C2+
C2-
VCC
V+
V-
T1IN
T2IN
R1OUT
R2OUT
GND
T1OUT
T2OUT
R1IN
R2IN
16
15
VCC+5V
0.1uF
0.1uF
L1
10uH
L2
10uH
DB9
XP3

7.4.3 单片机与 PC 机通信

在单片机与 PC 机通信时，双方要有一个约定好的协议，通信双方按协议交流，我们称这个协议为通信协议（communication protocols）。下面我们定义一个通信协议，然后根据协议我们进行编程。

指纹模块与 PC 机通信协议：

通信时采用半双工异步通讯，默认波特率 57600bps，可通过命令设置为 115200bps

传送的帧格式为 10 位，一个 0 电平起始位，8 位数据（低位在前）和一位停止位，无校验位

发送包，接收包格式：

包标志：1 个字节　　00H　命令包　　01H　数据包

地址码：2 个字节

包长度：1 个字节　　给出包内容的长度，单位为字节

包内容：最大长度为 256 字节

校验和：2 个字节　　校验方式为从发送包到包内容的各字节累加，高位溢出不管，取低两字节

应答信号：

81H　　接收正确

82H　　接收错，请求重发

83H　　接收错，终止当前数据包

84H　　主机忙

根据以上通信协议我们有下面的通信程序。

```
;--------------------------------------------------------------------------
;指纹模块与 PC 机通信程序
;撰写人：明 伟
;2002 年 1 月 7 日
;采用单片机串口进行通信
;通信时采用半双工异步通讯，默认波特率 57600bps，可通过命令设置为 115200bps
;传送的帧格式为 10 位，一个 0 电平起始位，8 位数据（低位在前）和一位停止位，无校验位
;P3.0----RXD
;P3.1----TXD
;定时器 2 设为波特率发生器晶振 11.0592MHz
$NOMOD51
;TONGXIN
$INCLUDE    (REG54.H)
NAME        TONGXIN
T2MOD  DATA     0C9H
PUBLIC  InitCom,GetPkgByte,SendPkgByte,GetPackage,SendPackage
        ORG     0000H
```

```
        JMP     MAIN
        ORG     000BH
        JMP T0_ROUTINE
        ORG     0100H
    MAIN:
        CALL    InitCom

        MOV       DPTR,#RESET

        CALL    SendPackage
VfyPwd0:
        MOV       DPTR,#VfyPwd
        CALL    SendPackage
        CJNE    R7,#1,VfyPwd0
    ;Password0:
        MOV       DPTR,#Password
        CALL    SendPackage
        MOV     DPL,#00H
        MOV     DPH,#20H
        CALL    GetPackage
        RET
;T0 timer routine
T0_ROUTINE:
        DJNZ    R0,ReCount
        JMP EXIT_T0
    ReCount:
        MOV     TMOD,#01H       ;定时 10ms
        MOV     TH0,#0DCH
        MOV     TL0,#00H
        SETB    EA
        SETB    ET0
        SETB    TR0

EXIT_T0:        RETI


;延时 50ms
    DELAY_50:
        MOV     R0,#255         ;等待溢出时间为 50ms
```

```
        MOV     TMOD,#01H  ;定时 10ms
        MOV     TH0,#0DCH
        MOV     TL0,#00H
        SETB    EA
        SETB    ET0
        SETB    TR0
        RET
;初始化串口为 57.6K 波特率
;?PR?InitCom?TONGXIN          SEGMENT    CODE
;   RSEG    ?PR?InitCom?TONGXIN
    InitCom:
        MOV     RCAP2H,#0FFH   ;定时器初值，波特率为 57600bps
        MOV       TH2,#0FFH
        MOV       RCAP2L,#0FAH
        MOV       TL2,#0FAH
        MOV       T2MOD,#0F0H  ;定时器 2 设为 16 位自动重装且向上溢出
        MOV       T2CON,#34H   ;定时器 2 设为波特率发生器
        MOV       SCON,#50H    ;将串口设置为方式 1,REN=1
        RET
;初始化串口为 115.2K 波特率
;?PR?InitComHigh?TONGXIN          SEGMENT    CODE
;   RSEG    ?PR?InitComHigh?TONGXIN
    InitComHigh:
        MOV     RCAP2H,#0FFH   ;定时器初值，波特率为 1152000bps
        MOV     RCAP2L,#0FDH
        MOV     T2MOD,#00H     ;定时器 2 设为 16 位自动重装且向上溢出
        MOV     T2CON,#34H     ;定时器 2 设为波特率发生器
        MOV     SCON,#50H      ;将串口设置为方式 1,REN=1
        RET
;-------------------------------------------------------------------------
;GetPkgByte (unsigned char   *Tranchar)
;参数：Tranchar 接收到的数据存放地址
;功能：从串口读入一个数据放到 Tranchar 中
;返回：1 数据读入成 功 0 数据读入失 败
;-------------------------------------------------------------------------
;接收子程序
;?PR?GetPkgByte?TONGXIN          SEGMENT    CODE
;   RSEG    ?PR?GetPkgByte?TONGXIN
    GetPkgByte:
```

```
        CLR RI
        MOV     A,SBUF
        MOVX    @DPTR,A                 ;此处今后还要修改
        RET
;-------------------------------------------------------------------------
;SendPkgByte (unsigned char    Tranchar)
;参数：Tranchar 要发送的数 据
;功能：从串口号发送一个数 据 Tranchar
;返回：1 数据发送成 功 0 数据发送失 败
;-------------------------------------------------------------------------
;?PR?SendPkgByte?TONGXIN         SEGMENT    CODE
;   RSEG    ?PR?SendPkgByte?TONGXIN
    SendPkgByte:
        CLR TI
        MOV     A,#0
        MOVC    A,@A+DPTR       ;此处今后还要修改
        MOV     SBUF,A
    HERE:   JNB TI,HERE
        RET
;-------------------------------------------------------------------------
;GetPackage (unsigned char *cmd)
;参数：cmd 包含串口收到各数据的结 构
;   INPUT   R7(no using)
;   OUTPUTR7
;using regester R0,R1,R2,R3,R4
;功能：从串口号读入一个包，包数据放在 cmd 结构中.
;此函数包含校验功能，并做出应答。当发送应答要求重发时，重新接收数据直至超过重发次
数
;或接收数据正确。
;返回：1 数据包读入成功 0 数据包读入失 败
;-------------------------------------------------------------------------
;?PR?GetPackage?TONGXIN          SEGMENT    CODE
;   RSEG    ?PR?GetPackage?TONGXIN
    GetPackage:
            MOV     R0,#3           ;symbol of error times
    ReGet:  MOV     R2,#0           ;累加和
;取包标志
    HERE0:  JNB RI,HERE0    ;等待中断
        CALL    GetPkgByte  ;从串口读入一个数据
```

```
        ADD     A,R2            ;累加
        MOV     R2,A            ;累加和放 R2 中
        INC     DPTR
;取地址码
        MOV     R3,#2
HERE1:
        JNB     RI,HERE1        ;等待中断
        CALL    GetPkgByte      ;从串口读入一个数据
        ADD     A,R2            ;累加
        MOV     R2,A            ;累加和放 R2 中
        INC     DPTR
        DJNZ    R3,HERE1
;取包长度
HERE2:
        JNB     RI,HERE2        ;等待中断
        CALL    GetPkgByte      ;从串口读入一个数据
        MOV     R3,A            ;包长度放 R7 中
        ADD     A,R2            ;累加
        MOV     R2,A            ;累加和放 R2 中
        INC     DPTR
;取包内容
        MOV     R1,#00H         ;放累加和高位
HERE3:
        JNB     RI,HERE3        ;等待中断
        CALL    GetPkgByte      ;从串口读入一个数据
        CLRC
        ADD     A,R2            ;累加
        MOV     R2,A            ;累加和放 R2 中
        MOV     A,#0
        ADDC    A,R1
        MOV     R1,A
        INC     DPTR
        DJNZ    R3,HERE3
;取校验和
HERE4:
        JNB     RI,HERE4        ;等待中断
        CALL    GetPkgByte      ;从串口读入一个数据
        MOV     R3,A            ;校验和高位
        INC DPTR
```

```
HERE5:
        JNB     RI,HERE5        ;等待中断
        CALL    GetPkgByte      ;从串口读入一个数据
        MOV     R4,A            ;校验和低位
        INC DPTR
;校验
JIAOYAN:
        MOV     A,R1            ;校验
        CLRC
        SUBB    A,R3
        CJNE    A,#0,SoutEot
        MOV     A,R2
        CLRC
        SUBB    A,R4
        CJNE    A,#0,SoutEot
;accept ture
GetTure:
        MOV     R7,#1           ;It is symbol of accept ture.
        CLRTI
        MOV     A,#81H
        MOV     SBUF,A
WAITE0:
        JNB     TI,WAITE0
        CLRTI
        MOV     A,#81H
        MOV     SBUF,A
WAITE1:
        JNB     TI,WAITE1
        JMP     EXIT
;Accept error
GetError:
        CLRTI
        MOV     A,#82H
        MOV     SBUF,A
WAITE2:
        JNB     TI,WAITE2
        CLR     TI
        MOV     A,#82H
        MOV     SBUF,A
```

```
WAITE3:
        JNB     TI,WAITE3
        JMP     ReGet
;Send end of information
SoutEot:
        DJNZ    R0,GetError   ;Send end of information
        MOV     R7,#0         ;It is symbol of accept EEROR.
        CLR TI
        MOV     A,#83H
        MOV     SBUF,A
WAITE4:
        JNB     TI,WAITE4
        CLR     TI
        MOV     A,#83H
        MOV     SBUF,A
WAITE5:
        JNB     TI,WAITE5
        JMP     EXIT

    EXIT:   RET
;---------------------------------------------------------------------------
;SendPackage (int comport SCMD scmd)
;参数：comport 串口号
;     INPUT   R7(no using)
;     OUTPUTR7
;using regiester R0,R1,R2,R3,R4,
;功能：从串口号发送一个包，包数据放结构 scmd 中，此函数自动发送校验，并等待应答,
;若应答要求重发，会自动重发直至下位机应答成功或要求中断。
;返回：1 数据包发送成功   0 数据包发送失败   2 模块忙
;---------------------------------------------------------------------------
;?PR?SendPackage?TONGXIN         SEGMENT   CODE
;     RSEG    ?PR?SendPackage?TONGXIN
SendPackage:
        MOV     R2,#0         ;累加和
;发包标志
        CALL    SendPkgByte   ;从串口发送一个数据
        ADD     A,R2          ;累加
        MOV     R2,A          ;累加和放 R2 中
        INC     DPTR
```

```
;发地址码
        MOV     R3,#2
HERE7:
        CALL    SendPkgByte  ;从串口发送一个数据
        ADD     A,R2         ;累加
        MOV     R2,A         ;累加和放 R2 中
        INC     DPTR
        DJNZ    R3,HERE7
;发包长度
        CALL    SendPkgByte  ;从串口发送一个数据
        MOV     R3,A
        ADD     A,R2         ;累加
        MOV     R2,A         ;累加和放 R2 中
        INC     DPTR
;发包内容
        MOV     R1,#00       ;放累加和高位
HERE9:
        CALL    SendPkgByte  ;从串口发送一个数据
        CLRC
        ADD     A,R2         ;累加
        MOV     R2,A         ;累加和放 R2 中
        MOV     A,#0
        ADDC    A,R1
        MOV     R1,A
        INC     DPTR
        DJNZ    R3,HERE9

;发校验和
        CLR     TI
        MOV     A,R1
        MOV     SBUF,A
HERE10:
        JNB     TI,HERE10    ;等待中断
        CLR     TI
        MOV     A,R2
        MOV     SBUF,A
HERE11:
        JNB     TI,HERE11    ;等待中断
;time-over program
```

```
        MOV     R0,#10          ;等待溢出时间为 100ms
        MOV     TMOD,#01H  ;定时 10ms
        MOV     TH0,#0DCH
        MOV     TL0,#00H
        SETB    EA
        SETB    ET0
        SETB    TR0

;accept respontion
HERE12:
        JNB     RI,R0_0
        CLR     ET0
        JMP     GETDATA
R0_0:
        CJNE    R0,#0,HERE12
        CLR     ET0
        JMP     ACKERROR
GETDATA:
        CLR     RI
        MOV     A,SBUF
        MOV     R1,A
HERE13:
        JNB     RI,HERE13
        CLR     RI
        MOV     A,SBUF
        CLRC
        SUBB    A,R1
        CJNE    A,#0,ACKERROR ;应答校验错
        CJNE    R1,#81H,NEXT0
        JMP     SendTure        ;send ture
NEXT0:
        CJNE    R1,#82H,NEXT1
        JMP     SendError       ;send error
NEXT1:
        CJNE    R1,#83H,NEXT2
        JMP     SendEOT         ;end of transimmition
NEXT2:
        CJNE    R1,#84H,SendEOT
        JMP     BUSY            ;model busy
```

```
SendTure:
        MOV     R7,#1           ;It is symbol of send ture.
        JMP     EXIT1
ACKERROR:
        MOV     R7,#0
        JMP     EXIT1
SendError:
        JMP     SendPackage  ;重发
SendEOT:
        MOV     R7,#0
        JMP     EXIT1
BUSY:
        MOV     R7,#2
EXIT1:  RET

_Reset:
        MOV       DPTR,#RESET
        CALL    SendPackage
        RET
    END
```

以上汇编程序也可以由下面 C 程序实现

```
/*-------------------------------------------------------------------------------------------------------*/
/*指纹模块与 PC 机通信程序                                                                               */
/*撰写人：明 伟                                                                                          */
/*2002 年 4 月 25 日                                                                                     */
/*采用单片机串口进行通信                                                                                 */
/*通信时采用半双工异步通讯，默认波特率 57600bps，可通过命令设置为 115200bps*/
/*传送的帧格式为 10 位，一个 0 电平起始位，8 位数据（低位在前）和一位停止位，无校验位*/
/*P3.0----RXD                                                                                            */
/*P3.1----TXD                                                                                            */
/*定时器 2 设为波特率发生器晶振 11.0592MHz                                                               */
/*-------------------------------------------------------------------------------------------------------*/
#include<SST89C54.H>
#define         uchar               unsigned    char
#define         uint                unsigned    int
#define         true        1
#define         false       0
#define         busy        2
```

```
uchar     timeover;             /*等待时间溢出标志*/
/*--------------------------------------------------------------------------*/
/*InitCom ()                                                                */
/*功能：初始化串口为 57.6K 波特率                                           */
/*--------------------------------------------------------------------------*/
void InitCom()
{
    RCAP2H=0xFF;          /*定时器初值，波特率为 57600bps          */
    RCAP2L=0xFA;
    T2MOD=0xF0;           /*定时器 2 设为 16 位自动重装且向上溢出      */
    T2CON=0x34;           /*定时器 2 设为波特率发生器          */
    SCON=0x50;        /*将串口设置为方式 1,REN=1          */
    return;
}
/*--------------------------------------------------------------------------*/
/*GetPkgByte (uchar   *Tranchar)                                            */
/*参数：Tranchar 接收到的数据存放地址                                       */
/*功能：从串口读入一个数据 放到 Tranchar 中                                 */
/*返回：  1   数据读入成功       0    数据读入失败                          */
/*--------------------------------------------------------------------------*/
uchar     GetPkgByte(uchar   Tranchar)
{
    if(RI==0)
        return     false;     /*数据读入失败*/
    RI=0;                 /*清除接收标志*/
    Tranchar=SBUF;        /*从串口读入一个数据 放到 Tranchar 中*/
    return     true;      /*数据读入成功*/
}

/*--------------------------------------------------------------------------*/
/*SendPkgByte (uchar *Tranchar)                                             */
/*参数：*Tranchar 要发送的数据                                              */
/*功能：从串口发送一个数据 *Tranchar                                        */
/*返回：1   数据发送成功       0    数据发送失败                          */
/*--------------------------------------------------------------------------*/
uchar     SendPkgByte (uchar Tranchar)
{
    TI=0;                 /*清除发送标志*/
    SBUF=Tranchar;        /*从串口发送一个数据 */
```

```
    if(TI==1)
    {
        TI=0;
        return    true; /*数据发送成功*/
    }
    return    false;          /*数据发送失败*/
}
/*-------------------------------------------------------------------------------------------------------*/
/*GetPackage    (CMD *cmd)                                                                              */
/*参数： cmd 包含串口收到各数据的结构                                                                   */
/*功能：   从串口读入一个包，包数据放在 cmd 结构中，此函数包含校验功能，并*/
/*做出应答。当发送应答要求重发时，重新接收数据直至超过重发次数或接收数据 */
/*正确。                                                                                                */
/*返回：1  数据包读入成功       0    数据包读入失败                                                   */
/*-------------------------------------------------------------------------------------------------------*/
uchar    GetPackage(uchar *cmd)
{
    uchar    i,j,k;
    uint  jyhi,jyhj;                /*校验和*/
    k=1;                            /*记录发送次数*/
    do
    {
        do{}while(RI!=1);                /*等待串口中断*/
        if(GetPkgByte(*cmd++)!=1)            /*读包标志*/
            return    false;
        jyhi=*cmd++;
    for(i=0;i<2;i++)
    {
        do{}while(RI!=1);
        if(GetPkgByte(*cmd)!=1)              /*读地址码*/
            return    false;
        jyhi+=*cmd++;
    }
    do{}while(RI!=1);
    if(GetPkgByte(*cmd)!=1)                  /*读包长度*/
        return    false;
    jyhi+=*cmd;
    j=*cmd++;
    for(i=0;i<j;i++)
```

```
    {
        do{}while(RI!=1);
        if(GetPkgByte(*cmd)!=1)                 /*读包内容*/
            return      false;
        jyhi+=*cmd++;
    }
    for(i=0;i<2;i++)
    {
        do{}while(RI!=1);
        if(GetPkgByte(*cmd++)!=1)       /*读校验和*/
            return      false;
    }
    /*校验*/

    jyhj=*--cmd+(*--cmd*256);
    if(jyhi==jyhj)
    {
        SendPkgByte(0x81);              /*数据包读入成功*/
        return      true;
    }
    else  if(k==4)
    {
        SendPkgByte(0x83);              /*数据包读入失败,中止当前数据包*/
        return      false;
    }
    else
    {
        SendPkgByte(0x82);              /*请求重发*/
        k++;
    }
    }while(1);
}
/*-------------------------------------------------------------------------------------------------------------*/
/*SendPackage     (SCMD scmd)                                                                                 */
/*参数：          scmd 待发送的数据                                                                           */
/*功能： 从串口发送一个包，包数据放结构 scmd 中，此函数自动发送校验，并等待应答,                              */
/*若应答要求重发，会自动重发直至下位机应答成功或要求中断 。                                                  */
/*返回：1   数据包发送成功        0     数据包发送失败      2   模块忙                                      */
/*-------------------------------------------------------------------------------------------------------------*/
```

```
uchar     SendPackage(uchar *scmd)
{
    uchar     i,j,k,ack;
    uint  jyh;
    k=1;                              /*记录发送次数*/
    do{
        SendPkgByte(*scmd);           /*发送包标志*/
        jyh=*scmd++;
        for(i=0;i<2;i++)
        {
            SendPkgByte(*scmd);       /*发送地址码*/
            jyh+=*scmd++;
        }
        SendPkgByte(*scmd);           /*发送包长度*/
        jyh+=*scmd;
        j=*scmd++;
        for(i=0;i<j;i++)
        {
            SendPkgByte(*scmd);       /*发送包内容*/
            jyh+=*scmd++;
        }
        *scmd++=jyh;
        *--scmd=jyh/256;
        for(i=0;i<2;i++)
            SendPkgByte(*scmd++);         /*发送校验和*/

        TMOD=0x01;                /*开中断，等待 100ms*/
        TL0=0xb0;
        TH0=0x3c;
        EA=1;
            ET0=1;
            TR0=1;

        do{
            if(timeover==0)
                return    false;          /*等待时间溢出*/
        }while(RI!=1);
        GetPkgByte(ack);          /*接收应答*/
        if(ack==0x81)             /*接收数据包正确*/
```

```
            return      true;
        else  if(ack==0x82)
        {               /*接收数据包错，重发*/
            if(k==4)  return false;
            k++;
        }
        else  if(ack==0x83)             /*接收数据包错，中止当前数据包*/
            return      false;
        else  if(ack==0x84)             /*模块忙*/
            return      busy;
    }while(1);
}
timer0() interrupt 1 using 1
{
    EA=0;
    ET0=0;
    TR0=0;
    timeover=0;                         /*等待时间溢出*/
    return ;
  }



void main(void)
{
    uchar           get[2];
    InitCom();
    SendPackage(Reset);                     /*复位*/
    SendPackage(VfyPwd);                    /*验证设备握手口令*/
    SendPackage(Password);
    GetPackage(get);
}
```

第 八 章 人 机 交 互

在单片机的控制系统的运行中，时常需要人为的干预才能保证它的正常运行，因此必须有一个人机交互的部分。键盘和显示器是人机交互的两个重要组成部件。

8.1 键盘及接口技术

8.1.1 键盘接口常见问题

键盘实质上是一组按键开关的集合。通常，按键所用开关为机械弹性开关，均利用了机械触点的合、断作用。一个电压信号通过机械触点的断开、闭合过程，其波形如图 8-1 所示。由于机械触点的弹性作用，一个按键开关在闭合时不会马上稳定的接通，在断开时也不会一下子断开，因而在闭合及断开的瞬间均伴随有一连串的抖动，抖动时间的长短由按键的机械特性决定，一般为 5~10ms，这是一个很重要的时间参数，在很多场合都要用到。

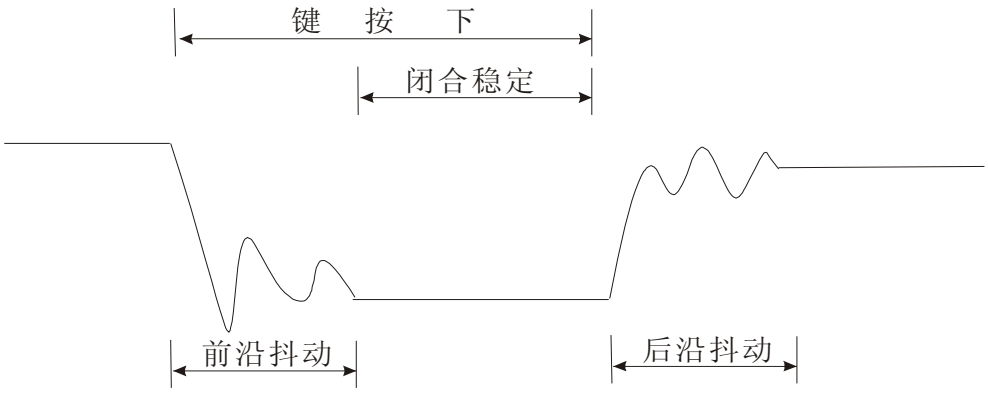


图 8-1 按键抖动信号波形

按键的稳定闭合期长短则是由操作人员的按键动作决定，一般为十分之几秒到几秒的时间，这个时间参数可作为一般的参考。

键的闭合与否，反应在电压上就是呈现出高电平或低电平，如果高电平表示断开的话，那么低电平则表示闭合，所以通过电平的高低状态的检测，便可确认按键按下与否。为了确保 CPU 对每一次按键动作只确认一次按键，必须消除抖动的影响。下面将介绍一些消除抖动的措施。

按键的抖动，通常有硬件、软件两种消除方法。

一、双稳态消抖动

双稳态消抖电路原理如图 8-2 所示。图中用两个与非门构成一个 RS 触发器。当按键未按下时，输出为 1 当键按下时，输出为 0。此时即使因按键的机械性能，使按键因弹性抖动而产生瞬时不闭合（抖动跳开 b），只要按键不返回原始状态 a，双稳态电路的状态不改变，输出保持为 0,不会产生抖动的波形。就是说即使 b 点的电压波形是抖动的，但经双稳态电路之后，其输出为正规的矩形方波，这一点很容易通过分析 RS 触发器的工作过程得到验证。

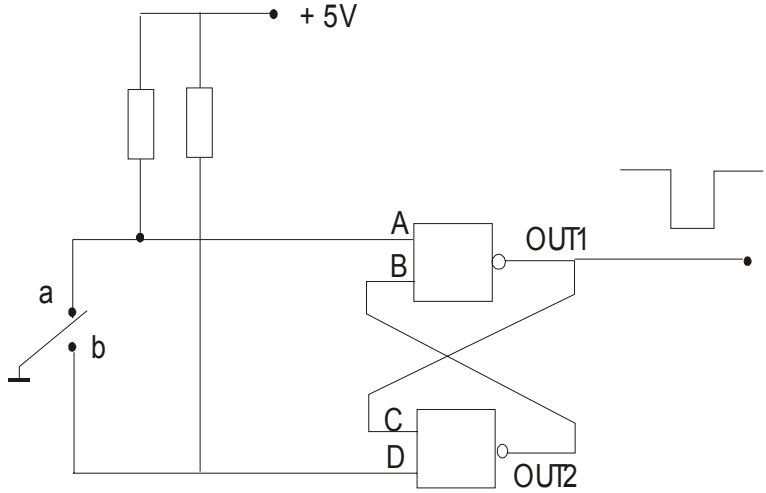


图 8-2 双稳态消抖电路

设开关 K 首先处于 a 位置。此时 RS 触发器的输出端 OUT1 为高电平 1，与非门 2 的输出为 OUT2=0，此输出引入到与非门 1 的一个输入端，将与非门 1 锁住，使其固定输出为 1。如果这时按动开关 K，即使开关 K 在 a 位置因弹性而瞬时抖动，在 a 处形成一连串抖动的波形，亦即 A 输入端出现一连串的 0 和 1，由于 B 端输入在 K 未到达 b 时始终为 0，所以无论 A 如何变化，OUT1 恒为 1。当 K 到达 b 时，一旦出现 D＝0，RS 触发器将出现状态的翻转，此时，OUT2＝1，导致 OUT1＝0，OUT1 又引回到门 2 的输入端，锁死门 2，让其输出恒为 1，即使 b 处的电压波形出现一连串的抖动，亦即 D 输入端出现一连串的 0 和 1，也不会影响 OUT2 的输出，因此 OUT1 也将恒为 0。如果将 RS 触发器当作一个黑匣子，只看输入与输出的波形，将会发现输出是消除了抖动的输入。同样，在松开按键的过程中，只要一接通 a，输出为 1，在接通 a 的过程中，即使产生弹性抖动而瞬间离开 a，只要开关不再与 b 发生接触，双稳态电路的输出将不会改变。

二、滤波消抖电路

因为 RC 积分电路具有吸收干扰脉冲的作用，所以只要选择好适当地时间常数，让按键抖动信号通过此滤波电路，便可消除抖动的影响，如图 8-3 所示。

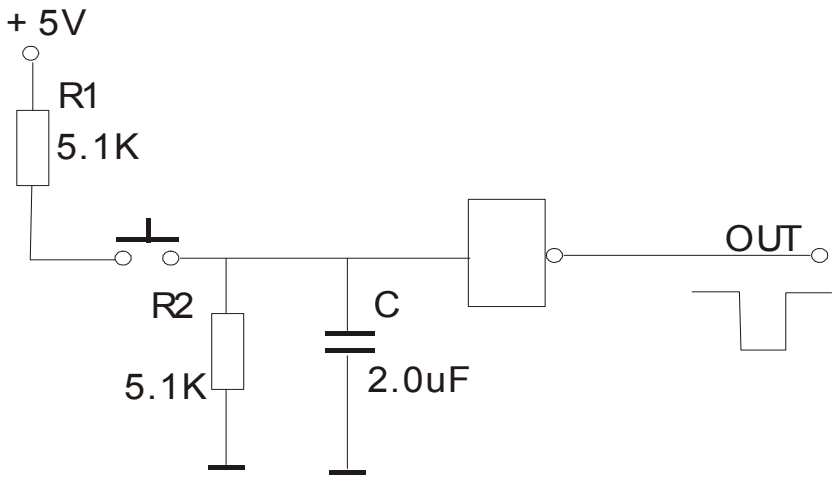


图 8-3 滤波消抖电路

当 K 未按下时，电容两端电压为 0,与非门输出为 1。当 K 按下时，由于 C 两端电压不能突变，即使在接触过程中出现抖动，只要 C 两端的充电电压波动不超过门的开启电压（TTL 为 0.8V 左右），门的输出将不会改变，这可通过适当选取 R1、R2 和 C 的值来实现。同样，K 在断开过程中，即使出现抖动，由于 C 两端电压不能突变，它要经过 R2 放电，只要 C 两端的放电电压波动不超过门的关闭电压，门的输出也不会改变。所以，关键在于 R1、R2 和 C 的时间常数的选取，必须保证 C 由稳态电压充电到开启电压或放电到关闭电压的延时大于或

等于 10ms。这既可计算确定，也可试验确定，图中参数仅供参考。

三、**软件消抖**

如果按键较多，硬件消抖将无法胜任，因此常采用软件的方法进行消抖。在第一次检测到有键按下时，执行一段延时 10ms 的子程序后再确认该键电平是否仍然保持闭合状态电平，如果保持闭合电平则确认真正有键按下，从而消除了抖动，后文的应用实例中就采用了软件消抖的方法。

8.1.2 矩阵式键盘接口设计

键盘可以分为独立连接式和矩阵（行列）式两类，实际上独立连接式只是矩阵式的一种特殊形式。因此我们只分析矩阵式键盘接口，独立连接式留给读者自行分析。

为了减少键盘与单片机接口时所占用 I/O 线的数目，在键数较多时，通常都将键盘排列成矩阵形式，如图 8-4 所示。

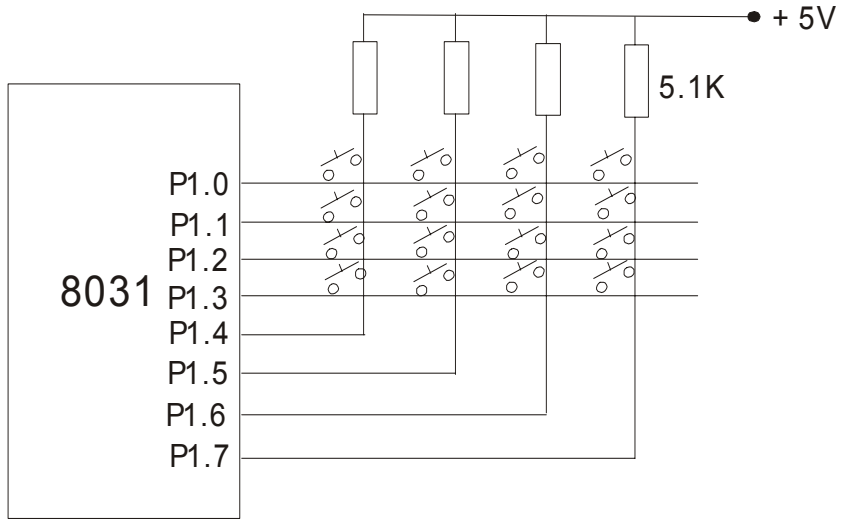


图 8-4　行列式键盘原理

每一水平线（行线）与垂直线（列线）的交叉处不相通，而是通过一个按键来连同。利用这种行列矩阵结构只需 N 条行线和 M 条列线，即可组成具有 N×M 个按键的键盘。

在这种行列矩阵式非编码键盘的单片机系统中，键盘处理程序首先执行等待按键并确认有无键按下的程序段，程序框图如图 8-5 所示。

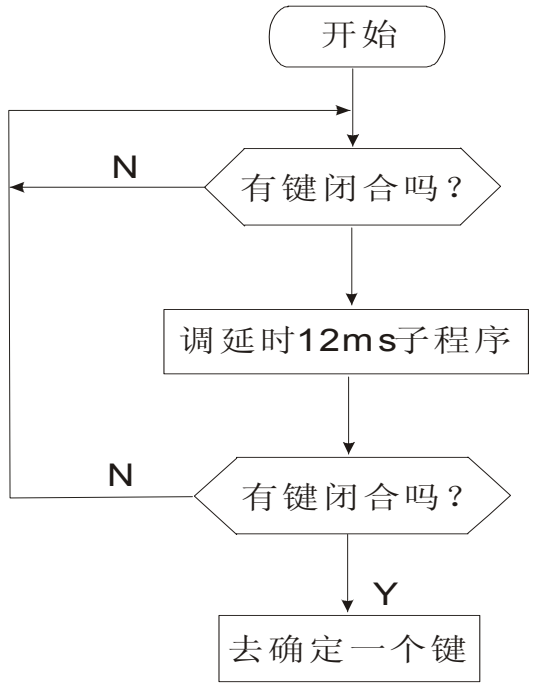


图 8-5　判有无键按下

当确认有按键按下后，下一步就要识别哪一个按键被按下。对键的识别通常有两种方法：一种是常用的逐行（或列）扫描查询法；另一种是速度较快的线反转法。

下面介绍行列扫描法的工作原理。

以图 8-4 所示的 4×4 键盘为例，说明扫描法识别哪一个按键被按下的工作原理。

首先判别键盘中有无键按下，由单片机 I/O 口向键盘送（输出）全扫描字，然后读入（输入）行线状态来判断。方法是：向列线（图中垂直线）输出全扫描字 00H，把全部列线置为低电平，然后将行线的电平状态读入累加器 A 中。如果有按键按下，总会有一根行线电平被拉至低电平，从而使输入不全为 1。

判断键盘中哪个键被按下是通过将列线逐列置低电平后，检查行输入状态实现的。方法是：依次给列线送低电平，然后查所有行线状态，如果全为 1,则所按下的键不在此列；如果不全为 1,则所按下的键必在此列，而且是在与零电平行线相交的交点上的那个键。

上面是线扫描法的描述，我们再来看一下线反转法是怎样工作的。

线反转法的操作主要有两步：

第一步：将行线编程为输入线，列线编程为输出线，并使输出线输出为全 0 电平，则行线中电平由高到低所在行为按键所在行。

第二步：同第一步完全相反，将行线编程为输出线，列线编程为输入线，并使输出线输出全为 0 电平，则列线中电平由高到低所在地列为按键所在列。

综合一、二两步的结果，可确定按键所在行和列，从而识别出所按地键。

8.1.3 键盘接口的应用

下面给出一个键盘的应用实例，以帮助大家熟悉新学的内容。

该键盘以 2×8=16 键的组合，我们以扫描查询法实现读键盘，下面给出读键盘的 C 程序。

```
/*============================================
            读键盘并返回值
============================================*/
uchar    keyword()
{
    uchar    i,scan[8]={0xfe,0xfd,0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
    KEY=0;
    Do
    {
        if(P1_0==0||P1_1==0)
        {
            delay(20);
            if(P1_0==0||P1_1==0)
            for(i=0;i<8;i++)
            {
                KEY=scan[i];
                if(P1_0==0)
                {
                    do{}while(P1_0==0);
                    return (i);
                }
                if(P1_1==0)
                {
                    do{}while(P1_1==0);
                    return (i+8);
                 }
             }
        }
    }while(1);
}
```

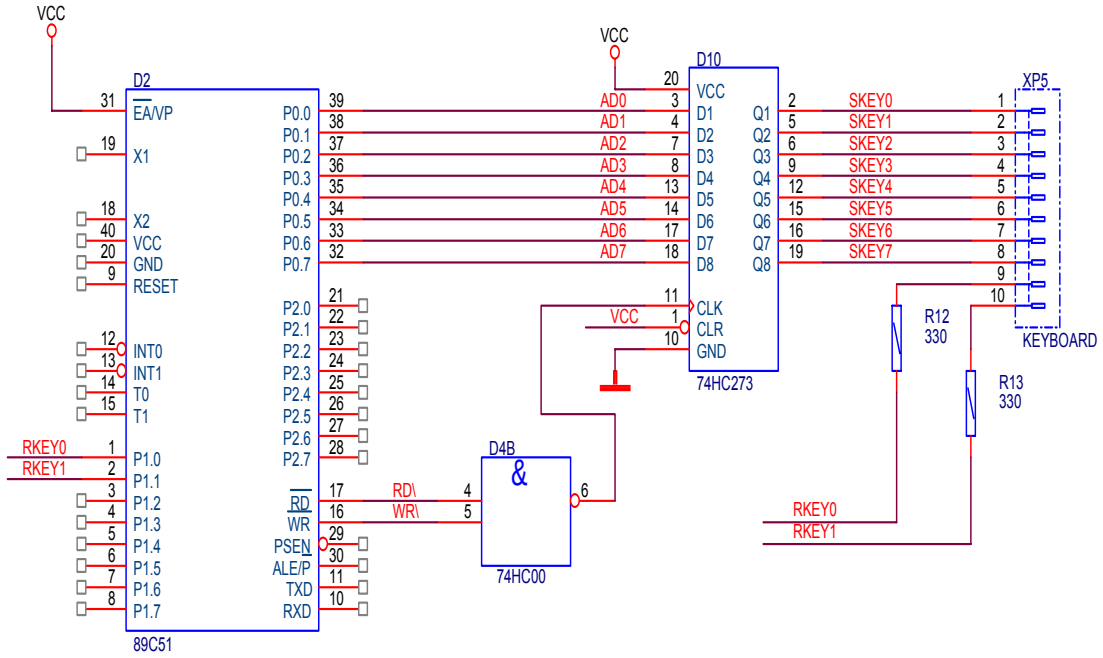


图 8-6 键盘电路硬件实图

```
/*================================================*/
/*返回键值                                        */
/*================================================*/
uchar   readkey()
{
    uchar   i;
    switch(i=keyword())
    {
        case     0: keynum=15;  break;
        case     1: keynum=14;  break;
        case     2: keynum=13;  break;
        case     3: keynum=12;  break;
        case     4: keynum=8;   break;
        case     5: keynum=5;   break;
        case     6: keynum=2;   break;
        case     7: keynum=10;  break;
        case     8: keynum=9;   break;
        case     9: keynum=6;   break;
        case    10: keynum=3;   break;
```

```
        case    11: keynum=11;  break;
        case    12: keynum=7;   break;
        case    13: keynum=4;   break;
        case    14: keynum=1;   break;
        case    15: keynum=0;   break;
    }
}
```

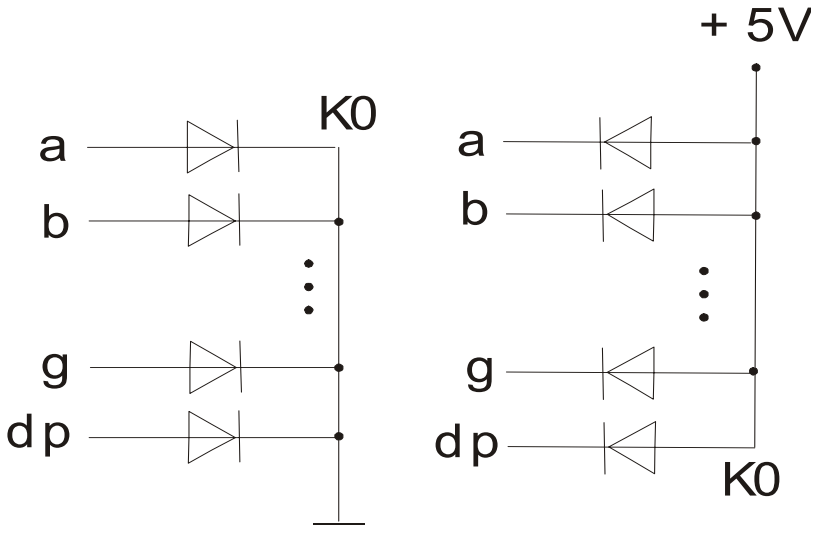
8.2 显示接口技术

在单片机的系统中，常用的显示器有：发光二极管显示器，简称 LED（Light Emitting Diode）；液晶显示器，简称 LCD（Liquid Crystal Display）；荧光管显示器。这三种显示器都有两种显示结构：段显示（7 段，“米”字型等）和点阵显示（5×7,5×8,8×8,128×64 点阵等）。而发光二极管显示又分为固定段显示和可以拼装的大型字段显示，此外还有共阳极和共阴极之分。

荧光管目前已极少使用，发光二极管由于生产简单，价格低廉和设计方便等优点，目前仍广泛应用于信号指示，路牌广告等场合。

8.2.1 LED 显示器

发光二极管的阳极连在一起的称为共阳极显示器，阴极连在一起的称为共阴极显示器。一位显示器通常由 8 个发光二极管组成，其中，7 个发光二极管构成字型“8”的各个笔划（段）a～g，另一个小数点为 dp 发光二极管。当在某段发光二极管上施加一定的正向电压时，该段笔划即亮；不施加电压则暗。为了保护 LED 不被损坏，需外加限流电阻。



(a) 共阴极 （b）共阳极

除上叙 7 段“8”字型显示器以外，还有 14 段“米”字型显示器和发光二极管排成 m×n 个点阵的显示器。其工作原理都相同，只是需要更多的 I/O 口线控制。

共阴极 7 段 LED 显示数字 0～F、文字、符号及小数点的编码（a 段为最低位，dp 点为最高位）如下表所示。

共阴极 7 段 LED 显示字型编码表

显示字符	共阴极段选码	显示字符	共阴极段选码
0	**3FH**	**C**	**39H**
1	**06H**	**D**	**5EH**
2	**5BH**	**E**	**79H**
3	**4FH**	**F**	**71H**
4	**66H**	**P**	**73H**
5	**6DH**	**U**	**3EH**
6	**7DH**	**T**	**31H**
7	**07H**	**Y**	**6EH**
8	**7FH**	**8**	**FFH**
9	**6FH**	**“灭”（黑）**	**00H**
A	**77H**		
B	**7CH**		

8.2.1.1 LED 静态显示方式

LED 显示器有静态显示和动态显示两种方式。

静态显示就是当显示器显示某个字符时，相应的段（发光二极管）恒定地导通或截止，直到显示另一个字符为止。例如，7 段显示器的 a，b，c 段恒定导通，其余段和小数点恒定截止时显示 7；当显示字符“8”时，显示器的 a，b，c，d，e，f，g 段恒定导通，dp 截止。

LED 显示器工作于静态显示方式时，各位的共阴极（公共端 K0）接地；若为共阳极(公共端 K0)则接+5V 电源。每位的段选线（a～dp）分别与一个 8 位锁存器的输出口相连，显示器中的各位相互独立，而且各位的显示字符一经确定，相应锁存的输出将维持不变。正因为如此，静态显示的亮度较高。这种显示方式编辑容易，管理也较简单，但占用 I/O 口线资源较多。因此，在显示位数较多的情况下，一般都采用动态显示方式。

8.2.1.2 LED 动态显示方式

在多位 LED 显示时，为了简化电路，降低成本，将所有位的段选线并联在一起，由一个 8 位 I/O 口控制。而共阴极（或共阳极）公共端 K0 分别由相应的 I/O 线控制，实现各位的分时选通，下图所示为 6 位共阴极 LED 动态显示接口电路。

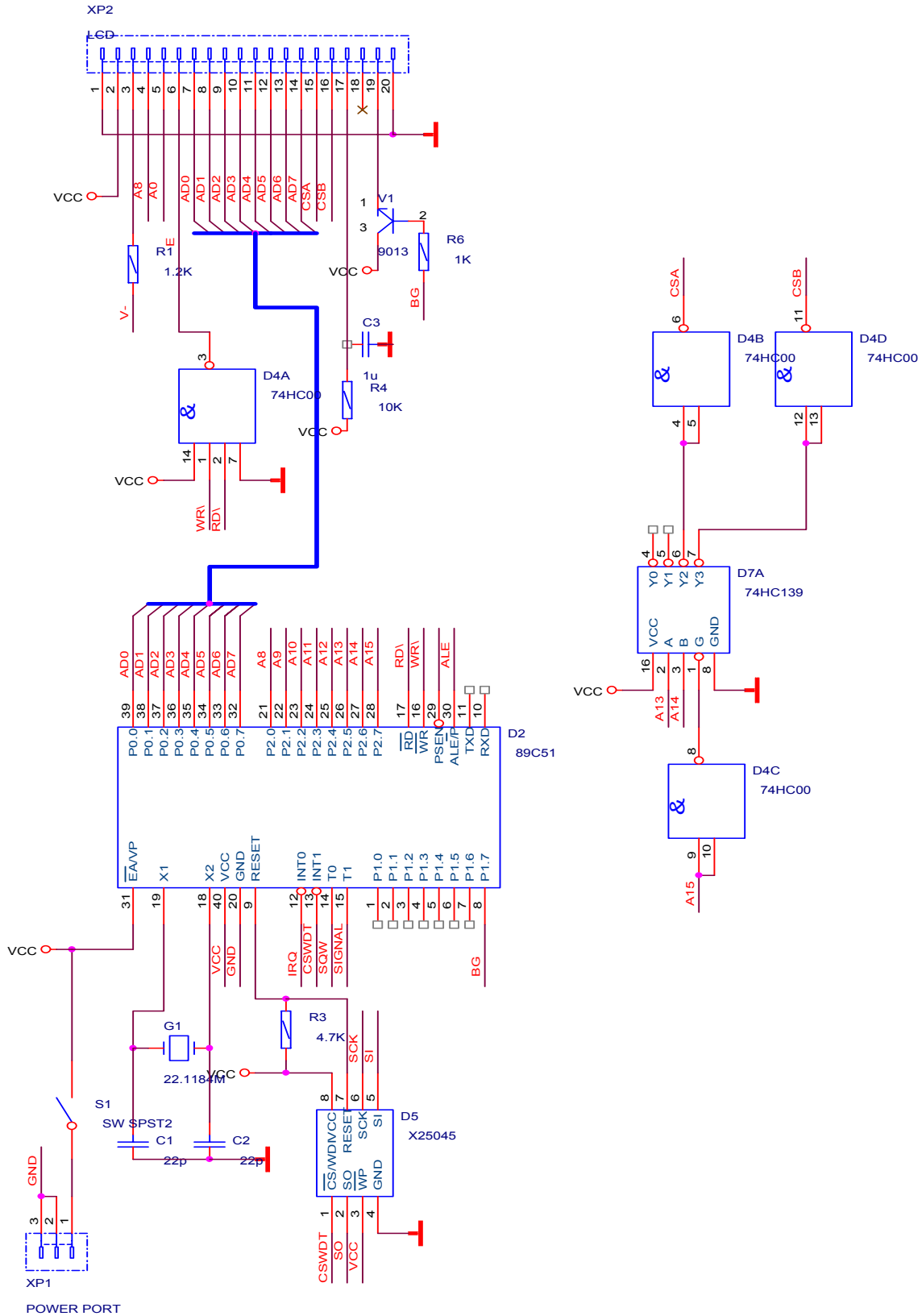
由于所有 6 位段选线皆由一个 I/O 口控制，因此，在每一瞬间，6 位 LED 会显示相同的

字符。要想每位显示不同的字符，就必须采用扫描法轮流点亮各位 LED，即在每一瞬间只使某一位显示字符。在此瞬间，段选控制 I/O 口输出相应字符，而位选则控制 I/O 口在该显示位送入选通电平，以保证该位显示相应字符。如此轮流，使每位分时显示该位应显示的字符。例如，要求显示“LL0－20”时，I/O1 和 I/O2 轮流送入段选码、位选码及显示状态。段选码、位选码每送入一次后延时 1ms，因人眼的视觉暂留时间为 0.1s（100ms），所以每位显示的间隔不能超过 20ms，并保持延时一段时间，以造成视觉暂留效果，给人看上去每个数码管总在亮。这种方式称为软件扫描显示。

8.2.2 液晶显示技术

液晶显示器 LCD 是一种极低功耗显示器，近年来由于价格下降和设计手段的提高，其开始在便携式产品中大显身手，目前极受生产厂家和设计商们推崇。点阵式液晶显示器由于其优良的性能，因而在便携式仪器仪表、低功耗、手持式设备等电子产品中获得了广泛的应用。厂家在提供 LCD 的同时也会提供予以配套的 LCM（液显控制模块），这使得硬件设计人员很方便的就完成了硬件接口问题。这里我们将以信利公司的 MCS－G12864DYSY－2N 显示控制模块为例，说明汉字显示的完整过程。本文在论叙时可以不涉及硬件原理图但为方便读者理解，我还是给出相关部分的原理图，应用于不同的硬件系统时，只需修改硬件地址即可。

以下是一个简单应用的硬件原理图：


XP2
LCD
VCC
R1
1.2K
V1
9013
R6
1K
C3
1u
R4
10K
D4A
74HC00
D4B
74HC00
D4D
74HC00
D7A
74HC139
D4C
74HC00
D2
89C51
R3
4.7K
G1
22.1184M
C1
22p
C2
22p
S1
SW SPST2
D5
X25045
XP1
POWER PORT
GND

一、 准备工作

准备工作是一个很重要的阶段，在完成准备之后，以后的工作可以说完全是一个纯粹的程序化问题。

1. MCS－G12864DYSY－2N

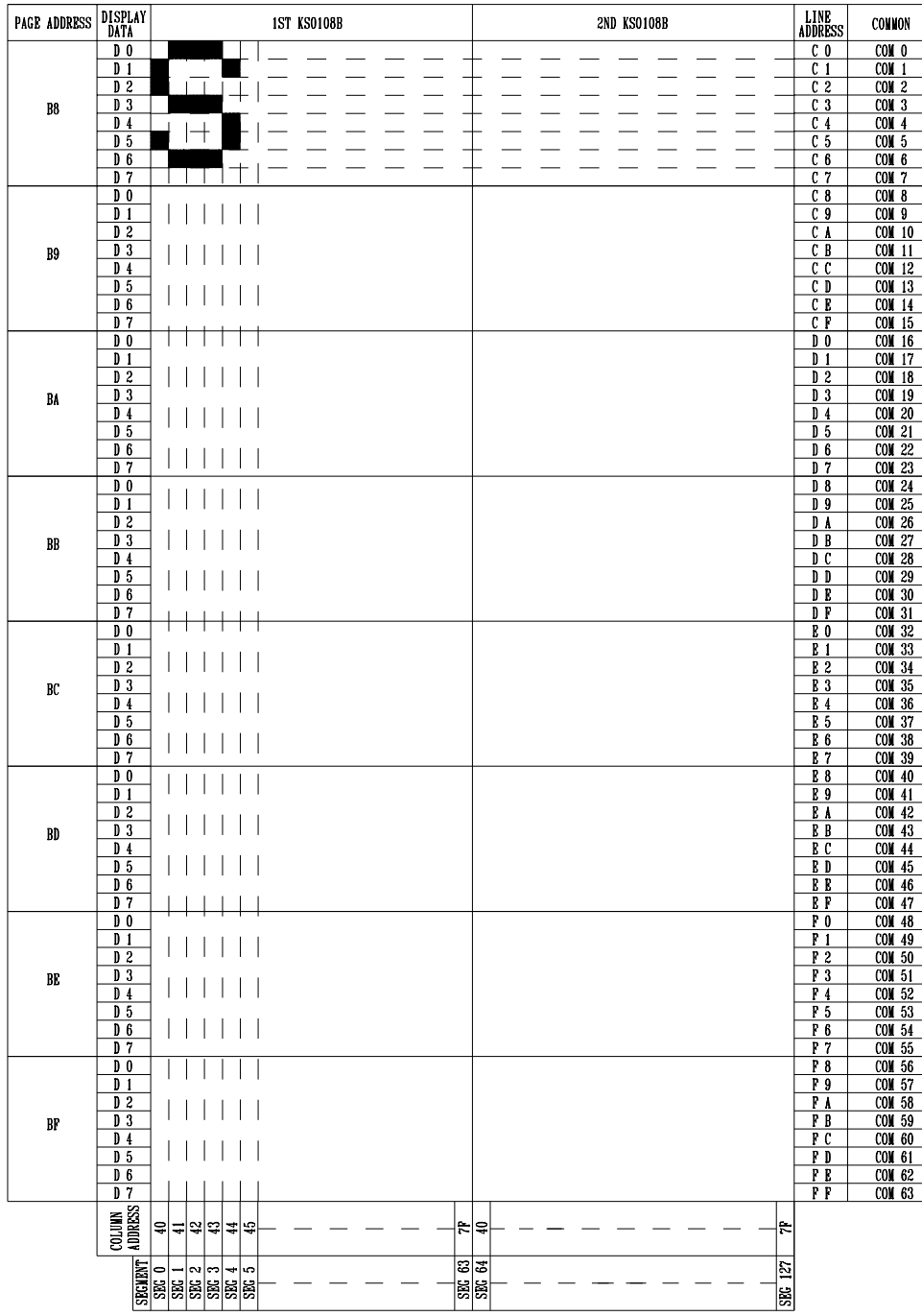
MCS－G12864DYSY－2N 是信利公司提供的液晶显示控制模块，共七条指令，结合 RS、R/W 引脚共同起作用，其指令见下表：

指 令	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	功 能
显示开/关	0	0	0	0	1	1	1	1	1	0/1	控制显示的开关.内部状态和显示 RAM 数据不受影响。. 0:关 1:开
设置 Y 地址	0	0	0	1	Y 地址 (0~63)						在 Y 地址寄存器中设置 Y 地址.
设置页地址 (X 地址)	0	0	1	0	1	1	1	页地址 (0~7)			在 X 地址寄存器中设置 X 地址.
显示开始行	0	0	1	1	显示开始行 (0~63)						指明将显示 RAM 中的数据显示在屏幕的顶端.
状态读	0	1	忙	0	开/关	复位	0	0	0	0	读状态 忙 0 ：读 1 ：工作中 开/关 0 ：显示开 1 ：显示关 复位 0 ：正常 1 ：复位
写显示数据	1	0	写数据								写数据到显示 RAM 中(DB0:7). 写完后 Y 地址会自动加 1.
读显示数据	1	1	读数据								将显示 RAM 中的数据读到数据总线上。

为了节省篇幅，上面的指令我将不做详细的解释，相信一般的工程技术人员应能看懂。

2. 显示内存数据地址

该 LCD 分两块显示，分别由两个模块控制，其显示内存数据地址图如下



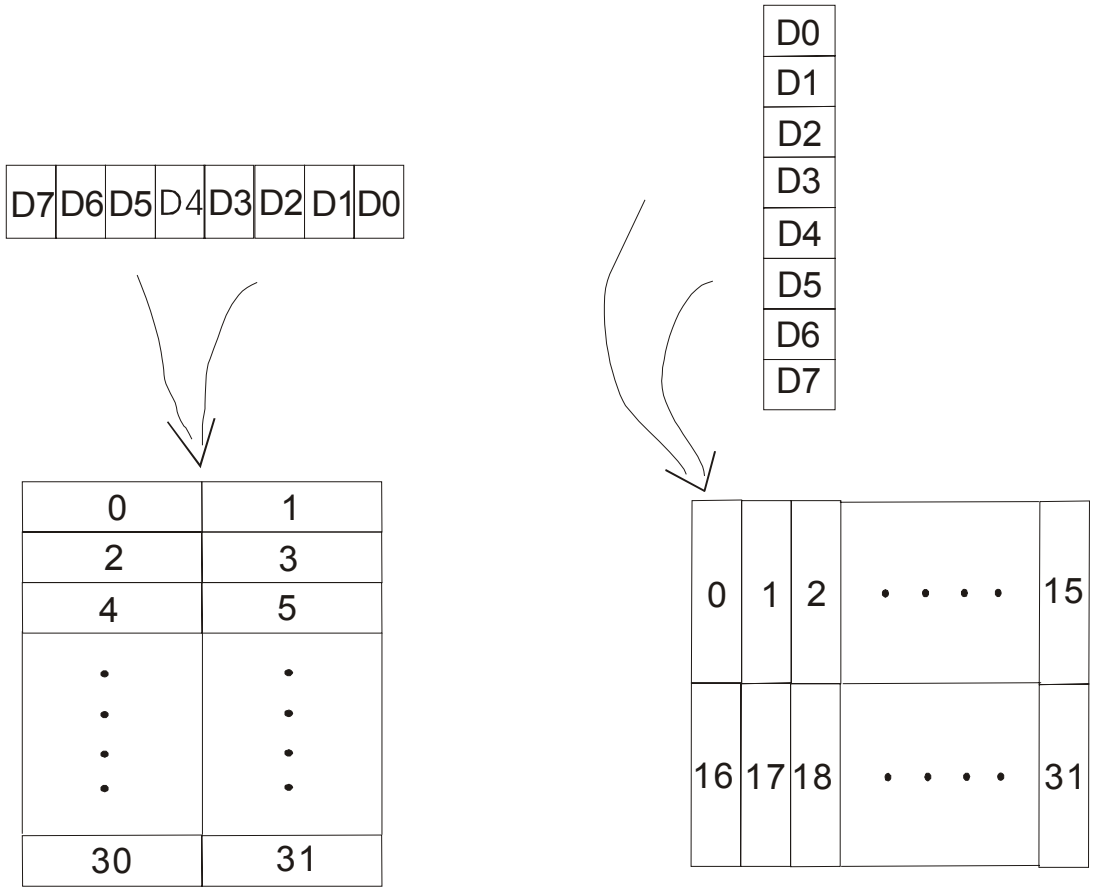
每写完一字节 Y 地址会自动加一。

从地址图中可以看出，其显示格式和我们平时所用的标准汉字库是不一样的。因而不能把现成的字库拿来使用，我们得自己做字库，这就是我们经常遇到的麻烦。

3． 字库转换

这里我给大家介绍一种字库转换方法，我叫它放缩字库转换法。标准的汉字库可以在 UCDOS 或天汇造字程序里找到。UCDOS 中是 HZK16，天汇中是 jbjd16，大小均为 256k。

常用汉字格式是一个以 bit 代表点的 16×16 矩阵，因而一个汉字占 32 字节。下面是标准汉字库和我们所用 LCD 显示格式的比较：



左边的为标准汉字库格式，右边的为 LCD 格式

从图中可以看出，不仅是每个字节里的每一 bit 的顺序不同了，而且每个字节在矩阵中的排序也有了变化。

有了以上比较，我们把每一位均扩大成一个字节，那么将形成一个 16×16 的矩阵，而我们现在的工作就可以明确为矩阵转换了。扩大前是以 bit 代表点，扩大后是以 byte 代表点。扩大的目的是为了方便转换，在完成转换后，我们又缩小，以 bit 代表点。

以下是我用放缩法实现的转换程序。

```
#include <stdio.h>
```

```
#include <conio.h>
#include <math.h>
void Zbyte(unsigned char *matx,unsigned char *new1) //一个字的转换程序
{                                                    //matx 指向待转换的 32 字节
   unsigned char f[16][16],s[16][16],w[32];
    //new1 指向新生成的 32 字节（转换过的）
   int i,j,k,m,Temp1,Temp2;                         //f，s 用于临时存放构建的矩阵
   Temp2=0;                                         // w[32]为新生成的 32 字节
   for(i=0;i<16;i++)                                //构建 16×16 的矩阵
   {
      for(j=0;j<16;)
      //取出每一位，放大成字节，方便转换
      {

        f[i][j]=*(matx+Temp2) & 0x80;                  //取出最高位，放大成字节
         j++;
        f[i][j]=*(matx+Temp2) & 0x40;                  //取出次高位，放大成字节
         j++;
        f[i][j]=*(matx+Temp2) & 0x20;
         j++;
        f[i][j]=*(matx+Temp2) & 0x10;
         j++;
        f[i][j]=*(matx+Temp2) & 0x8;
         j++;
        f[i][j]=*(matx+Temp2) & 0x4;
         j++;
        f[i][j]=*(matx+Temp2) & 0x2;
         j++;
        f[i][j]=*(matx+Temp2) & 0x1;
         j++;

        Temp2++;
    }
  }
   for(i=0;i<16;i++)
  //现在矩阵 f[16][16]是以字节代替 bit 的矩阵，我们对字节的操作，相当于我们对点阵
//中点的操作。在转换完后，原还未转换的 32 字节的第 0 字节的每一位，将作为转换后 32
//字节的前 8 字节的每个字节的第 0 位。因此，必须把 f[16][16]的前 8 字节中每个代表点
//的那位移到第 0 位，余下的要依此类推
```

```
    for(j=0;j<16;j++)

    {
      m=(j%8)-(i%8);
      f[i][j]=ldexp(f[i][j],m);
    }
//这里较难想清，建议读者应多做思考，完了再看下一步。

  for(i=0;i<16;i++)                          //矩阵转换实际是一个极费脑筋的事情，
                                             //我们要有足够的耐心
 for(j=0;j<16;j++)                          //我在做此转换时，每做一步，便画一矩阵
                                            //图进行观察，我觉得
      s[15-j][i]=f[i][j];
                              //此法较为有效。此处将行变为列，并在顺序上有所调整

  Temp1=0;
                              //现在将放大的位缩小还原，每 8 个字节合为一个字节
  for(i=0;i<16;i++)
  {

*(new1+Temp1)=s[i][0]|s[i][1]|s[i][2]|s[i][3]|s[i][4]|s[i][5]|s[i][6]|s[i][7];
   Temp1++;

*(new1+Temp1)=s[i][8]|s[i][9]|s[i][10]|s[i][11]|s[i][12]|s[i][13]|s[i][14]|s[i][
15];
   Temp1++;
  }

  for(i=0;i<32;i++)
  {
     w[0]=(*(new1+i)&0x01)<<7;
                          //由于显示格式不同，我们得将每个字节的位的高低进行调整
     w[1]=(*(new1+i)&0x02)<<5;                  //高低正好反向，这是由硬件决定的
     w[2]=(*(new1+i)&0x04)<<3;
     w[3]=(*(new1+i)&0x08)<<1;
      w[4]=(*(new1+i)&0x10)>>1;
      w[5]=(*(new1+i)&0x20)>>3;
      w[6]=(*(new1+i)&0x40)>>5;
      w[7]=(*(new1+i)&0x80)>>7;
```

```
        *(new1+i)=w[0]|w[1]|w[2]|w[3]|w[4]|w[5]|w[6]|w[7];
    }

    j=15;
    for(i=0;i<31;i=i+2)
    //由于显示的顺序，我们得对字节的顺序进行调整
    {
        w[j]=*(new1+i);
        j--;
    }
    j=31;
    for(i=1;i<32;i=i+2)
    {
        w[j]=*(new1+i);
        j--;
    }
    for(i=0;i<32;i++)
        *(new1+i)=w[i];
}

void main ()                                    //主函数主要用于读取和生成文件
{
 unsigned char matx[32],new1[32];
 FILE   *fp1,*fp2;
 fp2=fopen("d:\\tway\\font\\new1","wb");
 if((fp1=fopen("d:\\tway\\font\\gbjd16","rb"))==NULL)
 {
    puts("  can not open file . \n ");
    exit(1);
 }


 while(1)
 {
    fread(matx,sizeof(unsigned char),32,fp1);
    /*fseek(fp1,32*sizeof(unsigned char),SEEK_CUR);*/
    if(feof(fp1))
    {
```

```
        fclose(fp1);
        fclose(fp2);
        exit(1);
    }

    Zbyte(matx,new1);

    fwrite(new1,sizeof(unsigned char),32,fp2);
    /*fseek(fp2,32*sizeof(unsigned char),SEEK_CUR);*/
 }
}
```

→

转换过的字库应该和原字库一般大小，把它下载到存储器里就可以使用了。

二、 显示汉字

1、汉字寻址

要解决汉字显示，就得把你想要显示的汉字从 256k 字库中找出来，但如果你在程序代码区自建了小字库，那就另当别论了。

在 PC 机的文本文档中，汉字是以机内码的形式存储的，每个汉字占两个字节，分别由区码和位码组成。区码用于说明汉字所在的区，位码用于说明汉字所在区中的地址，汉字库每个区 94 个字，而汉字的区位码均从 A1H 开始，因此我们可以通过以下公式计算出每个汉字在汉字库中的位置。Address=32*[(区码－160－1)*94＋（位码－160－1）]，这样，在 Address 以后的 32 字节，就是您所要找的汉字的点阵了。这用 C 语言编程实现十分简单。

2、汉字显示

好了，完成以上的准备工作，我们的任务就已经完成一半了。剩下来的任务就只有编程了。以下是我的显示程序，都是已经实用化的，读者经过简单的修改，相信应能在自己的系统中使用。

```
/*==========================================================================*/
/* 对光标进行设置，我在整个显示屏上设置一个座标，光标的位置在（X，Y）处 */
/*1<=X<=8                                                                   */
/*1<=Y<=16                                                                  */
/* ==========================================================================*/
    void    setcursor(uchar x,uchar y)
```

```
    {
        if(y<9)
        {                           /*Y<9 时，在 CSA 控制的模块上显示*/
            busystate();            /*读忙状态*/
            CSAX=0xb7+1*x;          /*计算光标的 X 地址，CSAX  是硬件地址*/
            busystate();
            CSAY=0x38+8*y;
        }
        else
        {                               /*Y>9 时，在 CSB 控制的模块上显示*/
            busystate();
            CSBX=0xb7+1*x;
            busystate();
            CSBY=0x38+8*(y-8);
        }
    }
/*=============================================================================*/
/*显示一个汉字或字符在坐标（X,Y）处，Z=1 正显                                  */
/*显示汉字为 uchar   aword[]显示字符为 uchar     aword[]                      */
/*=============================================================================*/
    void   writeaword(uchar x,uchar y,uchar z)
    {
        uchar     i,j;
          j=*aword;
          /*取数组长，数组长放在数组的第一字节，汉字为 32,字符为 16*/

        setcursor(x,y);
        for(i=1;i<=j;i++)
        {
          if(i= =(j/2+1))           /*显示完汉字的上半部分再显示下半部分*/
              setcursor(x+1,y);
          if(y>8)
          {
              if(z= =0)
              {
                  busystate();
                  CSBWD=~*(aword+i);
                  /*对数组取反，用于反显，CSBWD 为硬件地址*/
              }
```

```
                    else
                    {
                        busystate();
                        CSBWD=*(aword+i);
                    }
                }
                else
                {
                    if(z==0)
                    {
                        busystate();
                        CSAWD=~*(aword+i);
                    }
                    else
                    {
                        busystate();
                        CSAWD=*(aword+i);
                    }
                }
            }
        }
```

自建小字库

有时我们并不需要把整个汉字库都用上，只需显示几条提示或命令，这样仅需几十个汉字就够用了。这时，我们可以把汉字库建在程序代码区，每个汉字声明成一个数组，譬如：

```
/*设*/ uchar     code
  she1[33]={32,0x40,0x40,0x42,0xCC,0x00,0x40,0xA0,0x9F,0x81,0x81,0x81,0x9F,0xA0,0x20,0x20,0x0
0, 0x00,0x00,0x00,0x7F,0xA0,0x90,0x40,0x43,0x2C,0x10,0x28,0x26,0x41,0xC0,0x40,0x00};
```

数组中的第一字节表示长度。

使用时，直接拿数组名当地址就可以了。但声明前必须先将 ROM 中的二进制数据转换成十六进制。这可以自己编程去实现，上面就是我实现的程序。

这仅仅是 SST 单片机的硬件描述,殊不知,它的最大特点是:万变灵活,你看它可以实现以下普通单片机不能达到的境界:

1. 6 个 UART 串口.
2. 单片用户程序空间可达 72K.超过了标准 51 的 64K 限制.
3. 单片数据存储空间也可以超过 64K.也超过了标准 51 的 64K 数据存储空间的限制.
4. 程序和数据存储可互补利用.程序剩下的空间均可作为数据存储.这个也打破了标准 51 的程序和数据存储必须分开的限制.
5. 在片仿真功能,比专用仿真器更多的特殊仿真功能,可做仿真器.
6. 客户端产品的升级功能,注意:区别于开发工程师用到的 ISP 在线下载,因为 ISP 只是在开发过程中省去一个编程器.而不适合已经卖出去的产品升级维护.

能用好 SST 的单片机可为你的产品省去许多的系统成本,维护成本.这便是高手与普通人的区别:普通人只能抄或改别人的软硬件,采用廉价的器件去抢现存的市场.最后品质无法保证,返修费用增加,信誉降低,结果不能长久存于业界.而高手采用优质器件,硬件成本是靠技术节省下来的.