P87LPC76x 单片机作为 I²C 总线主控器

(AN464— Using the P87LPC76x microcontroller as an I²C bus master)

摘要

本文首先回顾了 I²C 总线的基本概念和功能，包括主控器、被控器、数据传送、地址设置、传送格式以及被控器地址的使用。本文接着描述了 P87LPC76x 的 I²C 接口的硬件特性，包括总线控制寄存器，数据寄存器和配置寄存器以及定时器 I。最后，本文还给出了支持单个主控器的 ASM 程序范例，包括数据传送和接收，以及总线错误恢复的编程规范。

概述

P87LPC76x 是一种小封装、低成本、高性能的单片机，它采用 80C51 加速处理器结构，而且片内带有支持 I²C 总线的硬件接口。

PHILIPS 公司发展的 I²C 已成为总线标准，并且由 PHILIPS 公司申请了专利。I²C 总线可使芯片通过一条简单的双向两线总线直接进行通信。工程设计者可以很方便的使用众多的 CMOS 器件和片内带有 I²C 接口的双极型芯片进行工业数字控制、家电产品和通信设备等方面的设计。图 1 给出了一个典型的系统配置。

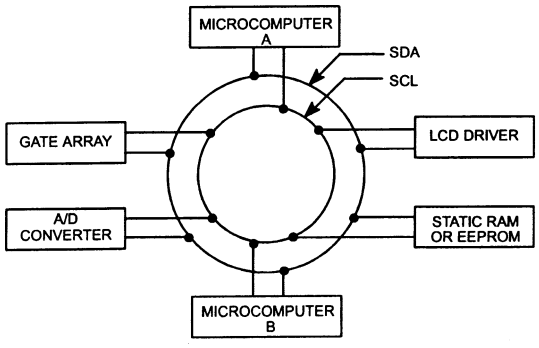

图 1 典型 I²C 总线的配置

I²C 基本系统的器件接口是非常简单的，因为这些器件可以直接用两条线相连接：一条串行数据线（SDA）和一条串行时钟线（SCL）。因为设计者不需要设计总线接口，而且方块草图上的功能模块对应于实际的芯片，所以设计者可以很快地完成从方块草图到最后方案的设计，修改和升级所设计的系统也很方便，只需从总线上卸下芯片或把芯片加到总线上。I²C 总线的简单性没有影响它的实用性，I²C 总线是一种可靠的、支持多主控器的总线，片内有地址定义功能和数据传送协议（参见图 2）。其它带有 I²C 总线的器件也是低成本和小封装的芯片。

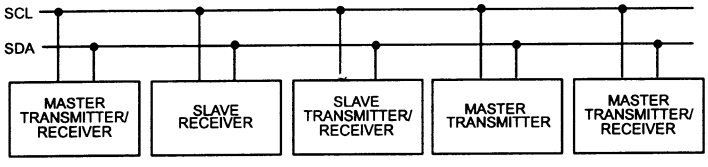

图 2 I²C 总线的连接

系统设计者利用 P87LPC76x 单片机设计系统将会是很有效率的。P87LPC76x 内部集成

的 I²C 总线协议使系统实现完全软件化定义，运用一些有用的软件模块库可以提高编程效率。另外，P87LPC76x 所支持的多主控器功能可以实现系统的快速测试，以及通过外接器件进行系统的校正。

P87LPC76x 可以加密 EPROM 程序。P87LPC76x 可以在 I²C 系统上作为主控器或被控器。除了有 I²C 接口外，P87LPC76x 还可以实现 I/O 和 RAM 的扩展，以及访问 EEPROM 和进行单片机之间的通信。

P87LPC76x I²C 总线多主控器功能也非常有用，但多数设计场合下并不需要运用这个功能。多数系统都运用单个主控器初始化器件之间的通信信息。本文详细介绍了如何使用 P87LPC76x 作为 I²C 总线的主控器。本文还介绍了 I²C 总线以及 P87LPC76x 单片机的 I²C 硬件接口。最后，本文还给出了一个关于 I²C 单个主控器通信的演示软件规范，这个软件规范十分通用，可以方便地把它移用到其它系统设计中。

本文中有关 I²C 总线的介绍不十分完整，有关 I²C 总线和 P87LPC76x 单片机的详细介绍可以参考相关资料。

I²C 总线

I²C 总线只有两条线：串行数据线（SDA）和串行时钟线（SCL），这两条线都通过一个上拉电阻与正电源连接，而且在总线不忙时保持高电平。总线上每个器件都有唯一的地址，无论它是单片机、LCD 驱动器、存储器，或者是键盘接口。根据器件的功能，这些器件可以在总线上作为发送器或接收器。产生信号或数据的器件为发送器；接收信号或数据的器件为接收器。显而易见，外围器件例如 LCD 驱动器只能作为接收器，而单片机或存储器可以发送和接收数据。

主控器和被控器

总线上的主控器和被控器都可以发送数据。主控器初始化数据传送格式和产生与传送数据相应的时钟信号；而被控器将被主控器寻址。注意，主控器可以作为发送器或接收器，在发送数据时作为发送器，在接收数据时作为接收器，在这两种情况下，主控器都要初始化数据传送格式。另一方面，被控器也可以作为发送器或接收器。

I²C 总线是一个多主控器的总线，I²C 总线系统中可以有多个主控器，这些主控器都可以初始化数据传送格式和控制总线（参见图 2）。一个单片机在一次数据传送过程中可以作为主控器；也可以在另一次数据传送过程中作为被控器，这时由总线上另一个单片机初始化数据传送格式。总线上主控器/被控器的关系不是固定的，在每次数据传送时都可能会改变。

当多个主控器连接到 I²C 总线上时，也许会同时占据总线，为了避免总线冲突和通信混淆，需要有总线仲裁。系统中各个器件以“线与”方式连接在总线上，这样 I²C 总线可以实现总线仲裁和时钟同步。在典型多主控器系统中，相应的单片机软件应该可以方便地使单片机器件在主控器和被控器模式之间变换，以及避免总线仲裁过程中的数据丢失。

数据传送

在每个时钟脉冲出现时，总线传送一个数据位（参见图 3）。在时钟信号高电平期间，SDA 线上的数据位应保持稳定，如果此时改变 SDA 线数据则被认为是总线的控制信号。SCL 线为高电平时，SDA 线电平由高至低的变化作为总线的起动条件；SCL 线为高电平时，SDA 线电平由低至高的变化作为总线的停止条件（参见图 4）。在起动条件和停止条件后的一小段时间之间，总线将处于“忙”状态。起动条件和停止条件由主控器产生。

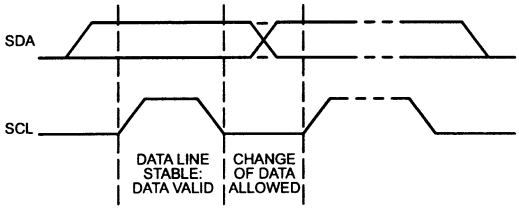

图 3 I^2C 总线上的位传送

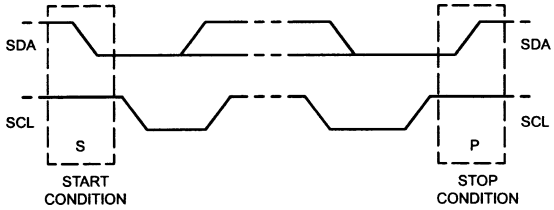

图 4 起动条件和停止条件

在起动和停止条件之间所传送的数据数量不受限制。传送数据的每个字节为 8 位，字节最高位数据先被传送，再依次传送其它位，最后传送一个应答位（参见图 5）。应答位的时钟脉冲由主控器产生，在出现与应答位对应的时钟脉冲时，产生应答位的器件将拉低 SDA 线，发送器则释放 SDA 线（参见图 6）。被控接收器在接收到每一个字节数据之后必须发送一个应答信号；而主控器在接收到被控发送器发送的数据后，也必须发送一个应答信号。如果接收器不能立即接收数据，它将拉低 SCL 线迫使发送器进入等待状态。设计 I^2C 系统时还需要考虑没有接收到应答信号的情况。例如：被寻址的器件正处于“忙”状态，而没有发送应答信号，这时主控器在定时器“溢出”后应产生停止条件，使总线进行其它的数据传送。这里的“其它的数据传送”是由同一个主控器或者多主控器系统中其它的主控器产生的。

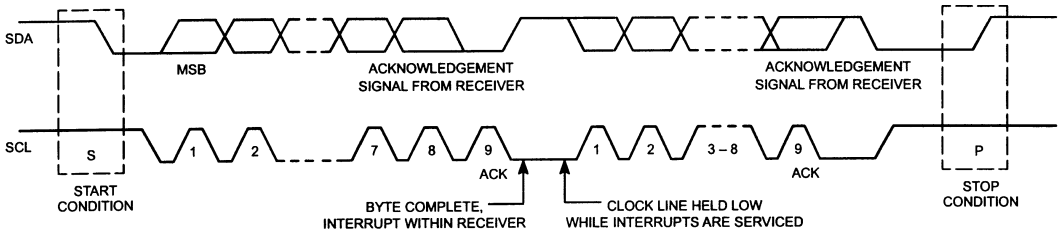

图 5 I^2C 总线上的数据传送

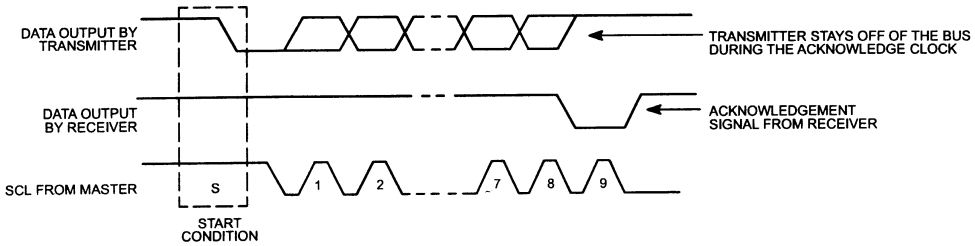

图 6 I^2C 总线上的应答位

关于“每个字节数据后产生应答位”，这里有两个例外情况。第一个例外是：当主控器作为接收器时，它必须在被控器发送完最后一个字节数据后产生非应答信号，与非应答信号相对应的时钟脉冲依然由主控器产生，但主控器并不拉低 SDA 线。我们把这种情况下的非

应答信号称之为“负应答位”。第二个例外是：被控器不再接收多余的字节数据时，它应该发送一个负应答位，器件发送数据而对方不接收时产生这种情况。

I²C 总线包含特殊的接口可用于与那些完全由软件模拟实现 I²C 功能的单片机以 I²C 总线方式通信，这称之为“低模式”。系统中所有的器件都内置 I²C 硬件接口时，可以避免“低模式”。

寻址和数据传送格式

总线上的每一个器件都有自己唯一的地址。在发送数据之前，主控器必须先发送被控器的地址，系统中与这个地址相匹配的被控器将产生一个应答信号。主控器在产生起动条件之后完成寻址。

总线上的器件地址为 7 位。地址字节中最后一位是方向位（R/W 位），R/W 为“0”表示主控器作为发送器（写操作）；R/W 为“1”表示主控器作为接收器（读操作）。图 7 中给出了一个完整的数据传送格式，由一个地址字节（写操作）和两个数据字节组成。

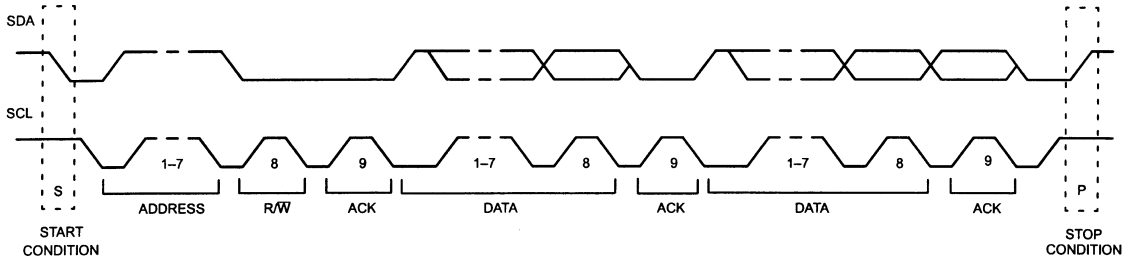

图 7 I²C 总线上的完整数据传送

起动条件产生后，主控器发送地址字节时，总线上每个器件将自己的地址与总线上的地址字节进行比较。如果某个器件的地址匹配，这个器件将传送一个应答信号，并可根据 R/W 位的值确定出它是被控接收器还是被控发送器。

I²C 总线上的每个器件节点都有唯一的 7 位地址。单片机的地址完全是可编程的，而外围器件的地址分为固定和可编程两部分。除了这里所讨论的“标准”地址外，I²C 总线协议允许使用广播地址和 CBUS 设备地址。

当主控器只与一个器件通信时，数据传送格式参见图 7，图中 R/W 位可以是任一数据传送方向。数据传送结束后，主控器将产生停止条件。如果主控器需要寻址系统中其它的器件，它可以产生一个新的起动条件，开始另一次数据发送或接收。

在主控器与几个不同的器件通信的情况下，将采用“重复起动”方式。在传送完最后一个字节数据（包括应答位或负应答位）之后主控器产生另一个起动条件，接着发送地址字节和数据，而不用产生停止条件。主控器与多个器件通信时可以使用读操作和写操作。在最后一个数据传送结束时，主控器产生停止条件并释放总线。相关的数据传送格式参见图 8。注意，产生重复起动条件时，可以变换所寻址的器件以及数据传送方向，而且不用释放总线。我们将从下文中了解到即使在主控器只与一个器件通信时变换数据传送方向也是很方便的。

在单个主控器系统中，利用重复起动条件结束每次数据传送比用一个停止条件和一个起动条件更有效率。在多主控器情况下，想确定出使用哪种数据格式最有效率将是很复杂的事情，因为当主控器使用重复起动模式时，它会长时间占用总线，这样将妨碍其它器件使用总线传送数据。

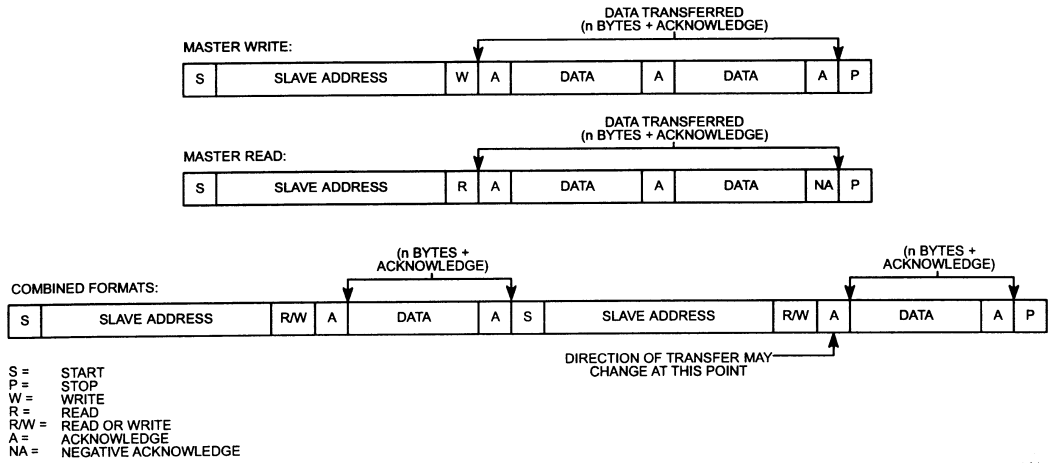

图 8 I^2C 总线数据格式

单元地址的使用

对于 I^2C 总线上的某些芯片，其唯一的总线地址是不足以解决有效的通信，这时需要使用芯片内部寻址机制。典型例子是当我们想访问被寻址存储器内部所指定的数据或者连续的存储器地址数据时，就需要使用存储器的单元地址。

典型的 I^2C 存储器芯片例如 PCF8570 片内有一个地址寄存器，这个寄存器在每一个字节数据被读或写后自动增益。当主控器与 PCF8570 连接通信时，主控器必须在发送被控器地址字节后接着发送单元地址，这个单元地址是主控器传送单个字节数据的存储器内部地址，或者是传送多个字节数据的连续存储器区域的起始地址。单元地址是为 8 位字节，与器件地址不同，单元地址不包含方向位（R/W）。与总线上所有字节数据传送格式一样，单元地址后必须跟随一个应答位。

存储器写数据操作格式参见图 9（a）。产生起动条件之后主控器发送被控器地址和方向位（这里为写操作模式，R/W=0）,接着发送单元地址和一些字节数据，以及产生停止条件。每次发送一个字节，同时存储器的地址寄存器自动产生增益。

存储器读数据操作格式（参见图 9（b））开始时与写数据操作格式一样：主控器发送被控器地址和方向位（R/W=0，为写模式），接着发送单元地址。为了改变数据传送方向，主控器产生一个重复起动条件，接着发送被控器地址和方向位（这时 R/W=1 为读模式）。存储器将发送从指定单元地址开始的字节数据到总线上，而且每个字节数据后跟随一个由主控器产生的应答位，但是最后一个字节数据后跟随一个负应答位，表明数据传送结束，最后由主控器产生的停止条件将结束读数据操作。

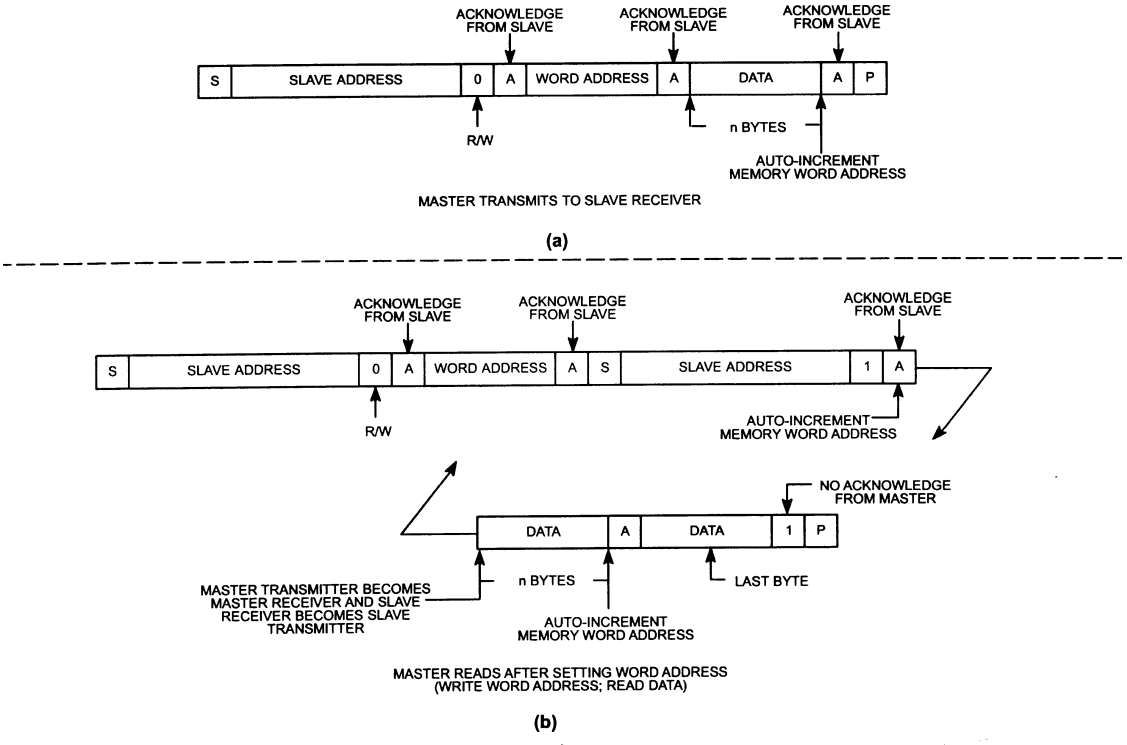

图 9 I²C 总线单元地址的使用

P87LPC76x I²C 硬件接口

P87LPC76x 系列单片机的 I²C 硬件接口可以高速运行，也可以简化系统通信的软件编程。 I²C 硬件接口激活和监视 SDA 线和 SCL 线，执行总线仲裁和检测总线状态错误，以及处理时钟延时和时钟同步。P87LPC76x I²C 硬件接口包括一个总线溢出定时器，即定时器 I。P87LPC76x I²C 硬件接口与软件同步运行，甚至在进入循环状态或产生中断时也同步运行。

当激活 I²C 总线时，P87LPC76x 端口 0 中 P0.0 与 P0.1 可以分别作为 SCL 和 SDA 线模拟行使 I²C 总线功能，端口 0 的管脚都有开漏输出。

P87LPC76x 五个中断源中的两个可用于 I²C 总线。设置中断使能寄存器中标志位 E12 使能 I²C 中断，而且在地址 033H 处开始运行 I²C 服务程序。在 SCL 线出现上升沿（表明总线上有新的数据）、产生起动条件、产生停止条件或者总线仲裁失败时，系统就会发生一个 I²C 中断请求（事先已使能中断）。标志位 ATN 的置位将导致 I²C 总线中断的发生（参见下文关于标志位 ATN 被置位的条件）。设置标志位 ETI 可以使能定时器 I 产生溢出中断，其服务程序在地址 73H 处开始执行。

I²C 总线由 3 个特殊功能寄存器控制。这 3 个寄存器为 I²C 控制寄存器（I2CON），I²C 配置寄存器（I2CFG）和 I²C 数据寄存器（I2DAT），它们的地址参见表 1。

表 1 I²C 总线的特殊功能寄存器地址

寄存器			位地址							
名称	符号	地址	最高位							最低位
I²C 控制寄存器	I2CON	D8	DF	DE	DD	DC	DB	DA	D9	D8
I²C 数据寄存器	I2DAT	D9	–	–	–	–	–	–	–	–
I²C 配置寄存器	I2CFG	C8	CF	CE	CD	CC	CB	CA	C9	C8

定时器 I

在 I²C 总线应用中，定时器 I 用于产生 I²C 总线时序和监视总线。在非 I²C 总线应用中，定时器 I 通常作为一个固定的时基。

定时器 I 用于产生 I²C 总线时序时，可以产生 I²C 时钟(SCL)信号。定时器 I 的时钟周期为一个机器周期（osc/12）,SCL 线的跳变速率是定时器 I 时钟频率的几倍。由于 P87LPC76x 单片机可以在宽范围振荡器频率下运行，因此需要调整 SCL 线频率为振荡器的某一分频。这样 I²C 总线依据系统振荡器频率可获得最佳的数据传输速率。设置特殊功能寄存器 I2CFG 中的两个标志位 CT0 和 CT1 可以调整 SCL 线的时序周期（参见表 2）。在定时器 I 中第 4 位翻转时，CT0 和 CT1 的数值将被装入定时器 I 的最后两位。当 CT0 和 CT1 被设置为“11”时，定时器 I 的第 3 位为“1”，其它情况下，第 3 位为“0”。当定时器 I 的第 4 位翻转时，SCL 线也跟着翻转。例如：CT1=1 且 CT0=0 时，定时器 I 最后 3 位被预置为“010”（即数值 2），这时定时器 I 的时间计数为 3、4、5、6、7、8（即 6 个时间计数或机器周期）。在计数为 8 时，定时器 I 的第四位将翻转，SCL 线也跟着翻转，而定时器 I 的最后 3 位又重新被预置为“010”。

表 2 CT1，CT0 数值定义

CT1,CT0	最小时间计数（机器周期）	CPU 时钟最大值（I²C 的频率为 100KHz）	溢出周期（机器周期）
1 0	7	8.4 MHz	1023
0 1	6	7.2 MHz	1022
0 0	5	6.0 MHz	1021
1 1	4	4.8 MHz	1020

定时器 I 可以作为“看门狗定时器”用于监视总线的“挂起”。在总线激活过程中，如果 SCL 线长时间停留在某一状态时，定时器 I 将产生溢出中断。当 SCL 线被钳制为低电平时，表明器件为无效的主控器或被控器；当 SCL 线被钳制为高电平时，表明器件无效，或者是因为外界对 I²C 总线的干扰导致所有的主控器退出总线仲裁。

定时器 I 产生溢出的时间是固定不变的，大约为 1024 个机器周期，其准确的数值由 CT0 和 CT1 的数值确定。激活 I²C 总线以及使能定时器 I 之后，在 SCL 线的下降沿产生时将复位定时器 I。如果 SCL 线在 1024 个机器周期中没有由高电平翻转为低电平，定时器 I 将计时溢出并产生中断。

定时器 I 溢出后，I²C 总线将复位，这对于多主控器系统的软件设计很有用。因为当发生总线错误时，总线将被“挂起”，软件做不出响应，而定时器 I 的溢出将释放 SCL 线，使多主控器系统中的其它器件可以继续使用 I²C 总线。

I2CON 寄存器

对 I²C 控制寄存器 I2CON 可以进行写操作（参见图 10）。编程者应该用下面程序范例中的位屏蔽方式对 I2CON 进行写操作。如果用位寻址方式去清除或设置 I2CON 寄存器，将可能会导致严重的后果。因为某些程序命令例如 CLR，先是读寄存器，再设置 I2CON 寄存器中的位，然后再写回，这时的写操作可能会影响寄存器的其它位。

图 10 I2CON 寄存器

读 I2CON

RDAT	ATN	DRDY	ARL	STR	STP	MASTER	—

RDAT 数据接收位。在 SCL 线上升沿时获取 SDA 线的值。与 I2DAT 中 RDAT 的数值相同，从 I2CON 中 RDAT 读数据的同时不清除 DRDY 和不释放 SCL 线。

ATN “注意”标志位。当 DRDY、ARL、STR 或 STP 中任意一个为“1”时，ATN 被置为“1”。检测 ATN 用于软件判断是否结束“等待循环状态”和是否总线上发生了某类事件。ATN 也可以激活 I^2C 中断请求。

DRDY “数据准备好”标志位。激活 I^2C 后，在 SCL 线上升沿时被设置，但对于空闲的被控器不设置 DRDY。读/写 I2DAT 或写 CDR 为“1”时清除 DRDY。

ARL 总线仲裁失败标志位。表明本器件试图控制 I^2C 总线时的仲裁失败。

STR 起动标志位。当检测到起动条件时被设置，但对于空闲被控器除外。

STP 停止标志位。当检测到停止条件时被设置，但对于空闲被控器除外。

MASTER 当本器件成为主控器时被设置。

写 I2CON

CXA	IDLE	CDR	CARL	CSTR	CSTP	XSTR	XSTP

CXA “清除数据传送状态”。写 CXA 为“1”清除数据传送状态。

IDLE 设置 IDLE 使被控器进入空闲模式，被控器在检测到下一个起动条件之前将忽略 I^2C 总线。如果软件设置 MASTRQ，空闲的被控器将变成主控器。

CDR 清除 DRDY。

CARL 清除 ARL。

CSTR 清除 STR。

CSTP 清除 STP。

XSTR “发送重复起动条件”。写 XSTR 为“1”可使总线产生重复起动条件。只有主控器可以写 XSTR。

XSTP “发送停止条件”。写 XSTP 为“1”时可使总线产生停止条件。

图 11 I2CFG 寄存器

SLAVEN	MASTRQ	CLRTI	TIRUN	—	—	CT1	CT0

SLAVEN 向此位写入“1”，使器件成为 I^2C 总线被控器。

MASTRQ 器件申请成为主控器时，向此位写入“1”。

CLRTI 向此位写入“1”将清除定时器 I 中断标志。当激活 I^2C 总线时，起动条件和停止条件的发送，以及数据的发送将清除 TIRUN。向此位写入“0”将清除和停止运行定时器 I。

CT1，CT0 根据系统振荡器频率设置这两位的值。CT1 和 CT0 的值决定 SCL 线高电平和低电平的最小时间。在不同的系统振荡器频率时，CT1 和 CT0 用于优化系统。

图 12 I2DAT 寄存器

读 I2DAT

RDAT	–	–	–	–	–	–	–

RDAT 接收数据位。在每个 SCL 线的上升沿时读取 SDA 线上的数据。读 I2DAT 将清除 DRDY 和发送激活状态，如果不想清除标志位，应该从 I2CON 中的 RDAT 读取数据。

写 I2DAT

XDAT	–	–	–	–	–	–	–

XDAT 发送数据位。下一个要传送的数据写入此位。写 XDAT 时将清除 DRDY 和设置发送激活状态。

I2CFG 寄存器

I^2C 配置寄存器 I2CFG 是一个可读/写的寄存器（参见图 11）。

I2DAT 寄存器

I^2C 数据寄存器 I2DAT 是一个可读/写的寄存器，其中最高位的数据是总线所传送的数据，其它 7 位被读时总为 0（参见图 12）。

数据传送激活状态

数据传送激活状态是 I^2C 总线内部的一种状态，这个状态受上文所描述的 I^2C 特殊寄存器的影响。当设置数据发送激活标志位时，I^2C 接口将拉低 SDA 线。对 I2DAT 写操作或设置 I2CON 中的 XSTR=1 或 XSTP=1 都可以设置数据发送激活状态。数据发送激活状态被设置时，标志位 ARL 将被设置为“1”，这样，在总线仲裁失败时，数据发送激活状态将自行复位。向 I2CON 中标志位 CXA 写入“1”或读 I2DAT 可清除数据发送激活状态。

编程范例

下面的程序可使 P87LPC76x 用于单主控器系统中作为 I^2C 总线的主控器。

单主控器系统的软件设计比多主控器系统简单些，因为软件设计者不用涉及变换主控器和被控器，也不用处理总线仲裁失败问题。

在单个主控器系统中，不需要用 I^2C 的状态起动中断，因此不需要使能 I^2C 中断。在相应程序被调用时，主控器初始化所有的数据传送格式。而在多主控器系统中，需要产生状态起动中断。在每一个 I^2C 状态时，程序中将用一个循环等待操作检测标志位 ATN，以及监视总线活动状态。因为 I^2C 总线可以在高速模式下工作，软件中循环等待操作只消耗一小段时间 ，而在软件中使用 I^2C 状态中断将会消耗较多的时间。

P87LPC76x 有一个字节寄存器硬件接口可以直接影响总线上器件的级别。软件与硬件寄存器一起构成 I^2C 总线协议。硬件和处理寄存器的服务程序是密不可分的。编程者在修改服务程序时一定要额外小心。

硬件系统复位后，主程序在地址 0 处（转到复位程序 Reset）开始执行。

主程序简单演示了 I^2C 总线的编程规范。主程序首先使能定时器 I 中断，再设置一些参数，这些参数用于读取被控器的数据。在本例子中，PCF8574A 8 位 I/O 口器件用作被控器，其中 0～3 口与按键相连接，4～7 口与 LED 相连接。RcvData 子程序用于读 I/O 口的数据。

当读取 PCF8574A 的数据字节时，按键数值也被保存并映象到相应的 LED 位。调用 SendData 子程序，可以把数据写到 PCF8574A I/O 口。SendData 和 RcvData 子程序可以相应地用于发送和接收多个字节数据，其数据数目由变量 ByteCnt 确定。

从 SendData 和 RcvData 子程序返回时，程序将测试标志 Retry 确定数据传送是否正确完成。如果数据传送没有正确完成，程序将重复数据传送操作。

在数据传送失败的情况下，程序将转回 MainLoop 处执行。

在复位程序之后是定时器 I 中断服务程序。如果定时器 I 溢出，系统将执行 73H 处指令，这时程序将使定时器 I 停止工作、清除定时器 I 中断和返回中断（开放其它中断），接着返回堆栈指针，再跳转到 Recover 子程序，试图恢复总线。

定时器 I 中断服务程序之后是 I²C 总线数据传送服务程序。其中 SendData、DcvData、SendSub 和 RcvSub 子程序由主程序调用，参量 XmtDat 的值作为数据传送缓冲区地址，用于保存所发送的数据。在本程序中 XmtDat 缓冲区长度为 4 个字节，可以根据系统需要确定数据缓冲区的大小。所接收的数据保存在 RcvDat 缓冲区中，RcvDat 缓冲区长度也为 4 个字节。以上 4 个程序都使用参量 SlvAdr 和 ByteCnt 相应地确定被控器地址和所传送的数据字节数。SendSub 和 RcvSub 子程序都使用参量 SubAdr 作为被控器的单元地址。

接下来的程序是一些子程序，由主程序调用处理 I²C 总线接口状态。在 SendAddr 子程序中调用 XmitAddr 子程序发送被控器地址。XmitByte 子程序用于发送数据到 I²C 总线上，也可用于发送单元地址。在发送完每一个字节数据后，这两个程序都将检测由被控器产生的应答信号。

RDAck 子程序调用 RcvByte 子程序读一个字节数据，然后发送一个应答信号给被控器。RDAck 子程序用于接收除最后一个字节数据外的所有数据（主控器将忽略最后一个字节的应答位）。RcvByte 子程序用于接收最后一个字节数据。

SendStop 子程序将产生一个 I²C 总线停止条件。RepStart 子程序用于发送一个重复起动条件，使主控器不用发送停止条件而直接开始下一个新的 I²C 总线数据传送状态。

地址发送

发送地址子程序用于发送地址字节，地址字节中最高位先被写入 I2DAT 中，接着其它 7 位依次被写入 I2DAT 中（使用 SendAd2 子程序）。SendAd2 子程序还清除起动条件，以便释放 SCL 线。当释放 SCL 线时，地址字节的最高位已经在 I2DAT 中。

数据接收

当接收数据时，程序应循环等待检测 ATN。在 ATN 等于 1 后，接着检测 DRDY，如果 DRDY 等于 1，则 SCL 线产生一个上升沿，这时可以从 I2CON 中的标志位 RDAT 或 I2DAT 中读取数据。

应该从 I2CON 中读取一个字节数据的最后一位，而不是 I2DAT（参见 RcvByte 程序的末尾），这样标志位 DRDY 不会被清除，SCL 的低电平周期将被延长。因为在接收最后一个数据位时，主控器必须响应一个应答信号，为了做到这一点，程序必须“等待”产生应答信号时钟（释放 SCL 线）时 SDA 出现高电平。SCL 线将保持低电平直到程序把应答信号写入 I2DAT 为止。

总线错误和非法状态

定时器 I 用于检测总线错误，非法状态由程序检测。定时器 I 溢出后，由 Recover 子程序恢复总线。Recover 子程序首先调用 FixBus 子程序检测是否只有 SDA 线被钳制为高电平，如果是这种情况，将发送额外时钟信号，强制产生总线停止条件；如果这种方法无效，将调

用 BusReset 子程序处理严重的总线错误。例如在时钟线被钳制为低电平的情况下，BusReset 子程序将置位 TIRUN，使定时器 I 运行，定时器 I 溢出后产生中断，这样就可以恢复严重的总线错误。

```
; *********************************************************************
;
;                    P87LPC764 I²C 总线单主控器编程规范
;
; 本程序演示了用单元地址接收 I²C 被控器的数据以及发送数据给被控器。
; *********************************************************************
$INCLUDE(MOD76x.ASM)          ; P87LPC76x 寄存器定义

; I²C 演示板的器件地址

LCD     EQU     72H           ; PCF8576 LCD 驱动控制器
LED7    EQU     76H           ; SAA1064 LED 驱动控制器
RTCLK   EQU     0A2H          ; PCF8563 时钟/日历芯片
RAM     EQU     0AEH          ; PCF8570 256 byte RAM
EEPROM  EQU     0A6H          ; 24WC02  256 byte EEPROM
DTMF    EQU     4AH           ; PCD3312 DTMF 发生器
PIO     EQU     4EH           ; PCF8574 8 位 I/O 口器件
KEYLED  EQU     7EH           ; PCF8574A I/O 口扩展器件
ADDAC   EQU     9EH           ; PCF8591 ADC 和 DAC 器件

; 数值定义
CTVAL   EQU     02H           ; 设置 I²C 寄存器标志位 CT1，CT0

; 屏蔽 I2CFG 标志位
BTIR    EQU     10H           ; 屏蔽 TIRUN
BMRQ    EQU     40H           ; 屏蔽 MASTRQ

; 屏蔽 I2CON 标志位
BCXA    EQU     80H           ; 屏蔽 CXA
BIDLE   EQU     40H           ; 屏蔽 IDLE
BCDR    EQU     20H           ; 屏蔽 CDR
BCARL   EQU     10H           ; 屏蔽 CARL
BCSTR   EQU     08H           ; 屏蔽 CSTR
BCSTP   EQU     04H           ; 屏蔽 CSTP
BXSTR   EQU     02H           ; 屏蔽 XSTR
BXSTP   EQU     01H           ; 屏蔽 XSTP

; 由 I²C 软件分配 RAM
BitCnt  DATA    21H           ; I²C 数据位计数器
ByteCnt DATA    22H
```

```
SlvAdr  DATA    23h         ; 被激活的被控器地址
SubAdr  DATA    24H

RcvDat  DATA    25H         ; 接收数据缓冲区（4 字节），地址为 25H～28H
XmtDat  DATA    29H         ; 发送数据缓冲区（4 字节），地址为 29H～2CH

StkSave DATA    2DH         ; 保存堆栈地址

Flags   DATA    20H         ; I²C 软件状态标志位
NoAck   BIT     Flags.0     ; 指明没有应答信号
Fault   BIT     Flags.1     ; 指明总线出错
retry   BIT     Flags.2     ; 指明 I²C 最后的数据传送失败，应该重复操作

; ********************************************************************
;                              程序开始
; ********************************************************************

; 复位和中断向量
        ORG     0000H
        AJMP    Reset       ; 复位向量

        ORG     0073H       ; 定时器 I 中断地址
TimerI: SETB    CLRTI       ; 清除定时器 I 中断
        CLR     TIRUN
        ACALL   ClrInt      ; 中断返回
        MOV     SP,StkSave  ; 恢复堆栈指针返回主程序
        AJMP    Recover     ; 试图恢复总线
ClrInt: RETI

        ORG     0100H

; ********************************************************************
;                          发送和接收数据程序
; ********************************************************************

; 发送数据字节给被控器
; 被控器地址在 SlvAdr,数据在 XmtDat 缓冲区中
; 被发送的数据字节数在 Bytecnt 中

Senddata:
        CLR     NoAck       ; 清除错误标志位
        CLR     Fault
        CLR     retry
        MOV     StkSave,SP  ; 为处理总线错误保存堆栈指针
```

```
        MOV     A,SlvAdr     ; 取被控器地址
        ACALL   SendAddr     ; 占据总线和发送被控器地址
        JB      NoAck,SDEX   ; 检测未送的应答信号
        JB      Fault,SDatErr ; 检测总线错误
        MOV     R0,#XmtDat   ; 设置数据传送缓冲区的头地址

SDLoop: MOV     A,@R0        ; 取数据
        INC     R0
        ACALL   XmitByte     ; 发送数据给被控器
        JB      NoAck,SDEX   ; 检测未送的应答信号
        JB      Fault,SDatErr ; 检测总线错误
        DJNZ    ByteCnt,SDLoop ; 这里 ByteCnt 为 1

SDEX:   ACALL   SendStop     ; 发送 I²C 停止信号
        RET

SDatErr:
        AJMP    Recover      ; 试图恢复总线

; 从被控器接收数据字节
; 被控器地址在 SlvAdr 中，数据字节数在 ByteCnt 中
; 数据返回到 RcvDat 缓冲区

Rcvdata:
        CLR     NoAck        ; 清除错误标志位
        CLR     Fault
        CLR     Retry
        MOV     StkSave,SP   ; 为处理总线错误保存堆栈指针
        MOV     A,SlvAdr     ; 取被控器地址
        SETB    ACC,0        ; 取总线读操作位
        ACALL   SendAddr     ; 发送被控器地址
        JB      NoAck,RDEX   ; 检测未送的应答信号
        JB      Fault,RDatErr ; 检测总线错误

        MOV     R0,#RcvDat   ; 设置数据接收缓冲区头地址
        DJNZ    ByteCnt,RDLoop ; 这里 ByteCnt 为 1
        SJMP    RDLast

RDLoop: ACALL   RDAck        ; 取数据和发送应答信号
        JB      Fault,RDatErr ; 检测总线错误
        MOV     @R0,A        ; 保存数据
        INC     R0
        DJNZ    ByteCnt,RDLoop ; 重复操作直到接收最后一个字节数据
```

```
RDLast: ACALL   RcvByte      ; 从被控器取最后一个数据
        JB      Fault,RDatErr ; 检测总线错误
        MOV     @R0,A        ; 保存数据

        MOV     I2DAT,#80H   ; 发送非应答信号
        JNB     ATN,$        ; 等待非应答信号发送
        JNB     DRDY,RDatErr ; 检测总线错误

RDEX:   ACALL   SendStop     ; 发送 I²C 总线停止信号
        RET

RDatErr:
        AJMP    Recover      ; 试图恢复总线

; 用单元地址发送数据给被控器
; 被控器地址在 ACC 中，单元地址在 SubAdr 中
; 所发送数据的字节数在 ByteCnt 中，发送的数据在 XmtDat 缓冲区中

SendSub:
        CLR     NoAck        ; 清除错误标志位
        CLR     Fault
        CLR     Retry
        MOV     StkSave,SP   ; 为处理总线错误保存堆栈指针
        MOV     A,SlvAdr     ; 取被控器地址
        ACALL   SendAddr     ; 占据总线和发送被控器地址
        JB      NoAck,SSEX   ; 检测未发送的应答信号
        JB      Fault,SSubErr ; 检测总线错误

        MOV     A,SubAdr     ; 取被控器单元地址
        ACALL   XmitByte     ; 发送单元地址
        JB      NoAck,SSEX   ; 检测未送的应答信号
        JB      Fault,SSubErr ; 检测总线错误
        MOV     R0,#XmtDat   ; 设置数据传送缓冲区头地址

SSLoop: MOV     A,@R0        ; 取数据
        INC     R0
        ACALL   XmitByte     ; 发送数据给被控器
        JB      NoAck,SSEX   ; 检测未送的应答信号
        JB      Fault,SSubErr ; 检测总线错误
        DJNZ    ByteCnt,SSLoop

SSEX:   ACALL   SendStop     ; 发送 I²C 停止条件
        RET
```

```
SSubErr:
        AJMP    Recover      ; 试图恢复总线

; 用单元地址从被控器接收数据
; 被控器地址在 SlvAdr 中，单元地址在 SubAdr 中
; 所接收的数据字节数在 ByteCnt 中，数据返回到 RcvDat 缓冲区

RcvSub: CLR     NoAck        ; 清除错误标志位
        CLR     Fault
        CLR     Retry
        MOV     StkSave,SP   ; 为处理总线错误保存堆栈指针
        MOV     A,SlvAdr     ; 取被控器地址
        ACALL   SendAddr     ; 发送被控器地址
        JB      NoAck,RSEX   ; 检测未送的应答信号
        JB      Fault,RSubErr ; 检测总线错误

        MOV     A,SubAdr     ; 取被控器单元地址
        ACALL   XmitByte     ; 发送单元地址
        JB      NoAck,RSEX   ; 检测未送的应答信号
        JB      Fault,RSubErr ; 检测总线错误

        ACALL   RepStart     ; 发送重复起始条件
        JB      Fault,RSubErr ; 检测总线错误
        MOV     A,SlvAdr     ; 取被控器地址
        SETB    ACC.0        ; 取总线读操作位
        ACALL   SendAd2      ; 发送被控器地址
        JB      NoAck,RSEX   ; 检测未送的应答信号
        JB      Fault,RSubErr ; 检测总线错误

        MOV     R0,#RcvDat   ; 设置数据接收缓冲区头地址
        DJNZ    ByteCnt,RSLoop ; 这里 ByteCnt 为 1
        SJMP    RSLast

RSLOOP:
        ACALL   RDAck        ; 取数据和发送应答信号
        JB      Fault,RSubErr ; 检测总线错误
        MOV     @R0,A        ; 保存数据
        INC     R0
        DJNZ    ByteCnt,RSLoop ; 重复操作直到接收最后一个字节数据

RSLast: ACALL   RcvByte      ; 从被控器取最后一个字节数据
        JB      Fault,RSubErr ; 检测总线错误
        MOV     @R0,A        ; 保存数据
```

```
        MOV     I2DAT,#80H  ; 发送非应答信号
        JNB     ATN,$       ; 等待
        JNB     DRDY,RSubErr ; 检测总线错误

RSEX:   ACALL   SendStop    ; 发送 I²C 停止条件
        RET

; 处理总线数据接收错误

RSubErr:
        AJMP    Recover ; 试图恢复总线

; ********************************************************************
;                                 子程序
; ********************************************************************

; 送地址字节
; 入口地址在 ACC

SendAddr:
        MOV     I2CFG,#BMRQ+BTIR+CTVAL ; 请求 I²C 总线
        JNB     ATN,$       ; 等待
        JNB     Master,SAErr ; 应成为总线上的主控器
SendAd2:
        MOV     I2DAT,A     ; 发送第一个数据位，清除DRDY
        MOV     I2CON,#BCARL+BCSTR+BCSTP ; 清除起动条件，释放 SCL 线
        ACALL   XmitAddr    ; 结束传送地址
        RET

SAErr:  SETB    Fault       ; 返回总线错误状态
        RET

; 字节传送程序
; 数据在 ACC 中
; XmitByte：传送 8 位数据
; XmitAddr：传送 7 位地址

XmitAddr:
        MOV     BitCnt,#8   ; 设置地址数值格式为 8 位
        SJMP    Xmbit2
XmitByte:
        MOV     BitCnt,#8   ; 设置数据格式为 8 位
Xmbit:  MOV     I2DAT,A     ; 发送数据位
```

```
Xmbit2: RL      A             ; 取下一个数据位
        JNB     ATN,$         ; 等待
        JNB     DRDY,XMErr    ; 数据准备好?
        DJNZ    BitCnt,Xmbit  ; 重复操作直到发送完所有的数据位
        MOV     I2CON,#BCDR+BCXA ; 转换为接收模式
        JNB     ATN,$         ; 等待应答位
        JNB     RDAT,XMBX     ; 是应答位?
        SETB    NoAck         ; 返回无应答信号状态
XMBX:   RET

XMErr:  SETB    Fault         ; 返回总线错误状态
        RET

; 字节数据接收程序
; RDAck：接收一个字节数据，然后发送应答位
; RcvByte：接收一个字节数据，数据返回到 ACC

RDAck:  ACALL   RcvByte       ; 接收一个字节数据
        MOV     I2DAT,#0      ; 发送应答位
        JNB     ATN,$         ; 等待应答位的传送
        JNB     DRDY,RdErr    ; 检测总线错误
        RET

RcvByte:
        MOV     bitCnt,#8     ; 设置数据位计数器
        CLR     A             ; 把数据初始化为 0
Rbit:   ORL     A,I2DAT       ; 取数据位，清除ATN
        RL      A             ; 左移数据
        JNB     ATN,$         ; 等待下一个数据位
        JNB     DRDY,RdErr    ; 数据准备好?
        DJNZ    BitCnt,Rbit   ; 重复操作直到接收了 7 个数据位
        MOV     C,RDAT        ; 取最后一个数据位，但不清除 ATN
        RLC     A             ; 形成完整字节数据
        RET

RdErr:  SETB    Fault         ; 返回总线错误状态
        RET

; 产生 I²C 停止条件程序

SendStop:
        CLR     MASTRQ        ; 取消主控器地位
        MOV     I2CON,#BCDR+BXSTP   ; 产生总线停止条件
        JNB     ATN,$         ; 等待 ATN
```

```
        MOV     I2CON,#BCDR ; 清除数据“准备好“标志位
        JNB     ATN,$       ; 等待传送停止条件
        MOV     I2CON,#BCARL+BCSTP+BCXA ; 释放 I²C 总线
        CLR     TIRUN       ; 停止定时器 I 的运行
        RET

; I²C 总线重复起动程序
; 入口地址在 ACC 中

RepStart:
        MOV     I2CON,#BCDR+BXSTR   ; 发送重复的起动条件
        JNB     ATN,$       ; 等待 ATN
        MOV     I2CON,#BCDR ; 清除数据“准备好”标志位
        JNB     ATN,$       ; 等待重复起动条件的传送
        MOV     I2CON,#BCARL+BCSTR  ; 清除起动标志位
        RET

; 总线错误恢复程序

Recover:
        ACALL   FixBus      ; 核查是否可以恢复总线
        JC      BusReset    ; 如果不能恢复，使用强制方式恢复总线
        SETB    Retry       ; 如果总线恢复完毕，返回到主程序
        CLR     Fault
        CLR     NoAck
        SETB    CLRTI
        SETB    TIRUN       ; 使能定时器 I
        SETB    ETI         ; 开放定时器 I 中断
        RET

; 下面的程序用于强制恢复总线，当 SCL 或 SDA 线被钳制时，
; 可以使用以下程序释放总线。（当定时器 I 溢出时，返回到 Recover 程序）

BusReset:
        CLR     MASTRQ      ; 释放总线
        MOV     I2CON,#0BCH ; 清除所有 I²C 标志位
        SETB    TIRUN
        SJMP    $           ; 等待定时器 I 溢出（这将使 I²C 硬件接口复位）

; 下面的程序试图在发生总线错误后重新控制 I²C 总线
; 如果成功，将返回进位标志位为 0;
; 如果失败，将返回进位标志位为 1

FixBus:
```

```
        CLR     MASTRQ          ; 关闭 I²C 功能
        SETB    C
        SETB    SCL             ; 确保 I/O 口没有钳制住 I²C 总线
        SETB    SDA
        JNB     SCL,FixBusEx    ; 如果 SCL 线为低电平，将不能恢复总线
        JB      SDA,RStop       ; 如果 SCL 和 SDA 线都为高电平，强制产生停止条件
        MOV     BitCnt,#9       ; 设置 BitCnt 为 9，试图释放总线
ChekLoop:
        CLR     SCL             ; 强制产生一个 I²C 时钟
        ACALL   SDelay
        JB      SDA,RStop       ; SDA 线为低电平?
        SETB    SCL
        ACALL   SDelay
        DJNZ    BitCnt,ChekLoop ; 重复产生时钟直到 SDA 线为低电平
        SJMP0   FixBusEx        ; 用这种方法失败则返回

RStop:  CLR     SDA             ; 由于 SCL 和 SDA 线都为高电平，用本程序强制
        ACALL   SDelay          ; 产生停止条件
        SETB    SCL
        ACALL   SDelay
        SETB    SDA
        ACALL   SDelay
        JNB     SCL,FixBusEx    ; SCL 和 SDA 线都还为高电平? 若是，则返回
        JNB     SDA,FixBusEx    ; 进位标志位为 0，表明总线已经恢复
        CLR     C
FixBusEx:
        RET

; 短时间延时程序（10 个机器周期）

SDelay: NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

; ********************************************************************
;                                 主程序
; ********************************************************************

Reset:  SETB    ETI             ; 开放定时器 I 中断
```

```
        SETB    EA              ；开放系统所有中断

；以下程序使用 I²C 演示板进行测试

MainLoop:
        MOV     SlvAdr,#KEYLED ；设置被控器地址（8 位 I/O 口器件）
        MOV     ByteCnt,#1H ；设置字节计数器
        MOV     SubAdr,#0H  ；设置被控器单元地址
        ACALL   RcvSub      ；从被控器处读数据
        JB      Retry,MainLoop  ；如果出现问题，则重复操作

        MOV     A,RcvDat    ；取所要接收的数据字节
        ANL     A,#0FH      ；屏蔽按键数据位
        SWAP    A           ；把按键数据位映象到 LED
        ORL     A,#0FH      ；不锁住按键数据位
        MOV     XmtDat,A    ；返回数据

ML2:    MOV     SlvAdr,#KEYLED ；设置被控器地址（8 位 I/O 口器件）
        MOV     ByteCnt,#1H  ；设置字节计数器
        MOV     SubAdr,#0H  ；设置被控器单元地址
        ACALL   SendSub     ；发送数据给被控器
        JB      Retry,ML2   ；如果出现问题，则重复操作

        SJMP    MainLoop    ；重复操作把按键数据位映象到 LED
        ; ================================================================
        ORG     0FD00H      ；EPROM 配置字节（UCFG1）
        DB      038H        ；关闭看门狗 WDT，系统复位后端口线置为高电平
                            ；掉电复位电压等于 2.5V,CLK=1，时钟频率为外部
                            ；晶振频率的两倍，振荡器类型为高频晶振
        ;
        END
```