目 录

第1章 ZLG7290B 简介

1.1 主要特性

- 直接驱动 8 位共阴式数码管（1 英寸以下）或 64 只独立的 LED；
- 能够管理多达 64 只按键，自动消除抖动，其中有 8 只可以作为功能键使用；
- 段电流可达 20mA，位电流可达 100mA 以上；
- 利用功率电路可以方便地驱动 1 英寸以上的大型数码管；
- 具有闪烁、段点亮、段熄灭、功能键、连击键计数等强大功能；
- 提供有 10 种数字和 21 种字母的译码显示功能，或者直接向显示缓存写入显示数据；
- 不接数码管而仅使用键盘管理功能时，工作电流可降至 1mA；
- 与微控制器之间采用 I^2C 串行总线接口，只需两根信号线，节省 I/O 资源；
- 工作电压范围：＋3.3～5.5V；
- 工作温度范围：－40～＋85℃；
- 封装：DIP-24（窄体），SOP-24。

1.2 描述

ZLG7290B 是广州周立功单片机发展有限公司自行设计的数码管显示驱动及键盘扫描管理芯片。能够直接驱动 8 位共阴式数码管（或 64 只独立的 LED），同时还可以扫描管理多达 64 只按键。其中有 8 只按键还可以作为功能键使用，就像电脑键盘上的 Ctrl、Shift、Alt 键一样。另外 ZLG7290B 内部还设置有连击计数器，能够使某键按下后不松手而连续有效。采用 I^2C 总线方式，与微控制器的接口仅需两根信号线。该芯片为工业级芯片，抗干扰能力强，在工业测控中已有大量应用。

1.3 引脚图

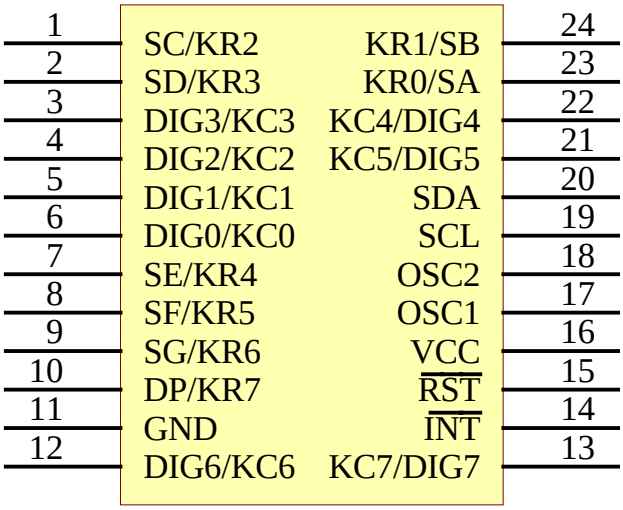

图 1.1 ZLG7290B 引脚图（DIP-24，SOP-24）

第2章 引脚功能说明

表 2.1 ZLG7290B 引脚功能表

引脚序号	引脚名称	功能描述
1	SC/KR2	数码管 c 段 / 键盘行信号 2
2	SD/KR3	数码管 d 段 / 键盘行信号 3
3	DIG3/KC3	数码管位选信号 3 / 键盘列信号 3
4	DIG2/KC2	数码管位选信号 2 / 键盘列信号 2
5	DIG1/KC1	数码管位选信号 1 / 键盘列信号 1
6	DIG0/KC0	数码管位选信号 0 / 键盘列信号 0
7	SE/KR4	数码管 e 段 / 键盘行信号 4
8	SF/KR5	数码管 f 段 / 键盘行信号 5
9	SG/KR6	数码管 g 段 / 键盘行信号 6
10	DP/KR7	数码管 dp 段 / 键盘行信号 7
11	GND	接地
12	DIG6/KC6	数码管位选信号 6 / 键盘列信号 6
13	DIG7/KC7	数码管位选信号 7 / 键盘列信号 7
14	$\overline{\text{INT}}$	键盘中断请求信号，低电平（下降沿）有效
15	$\overline{\text{RST}}$	复位信号，低电平有效
16	Vcc	电源，＋3.3～5.5V
17	OSC1	晶振输入信号
18	OSC2	晶振输出信号
19	SCL	I^2C 总线时钟信号
20	SDA	I^2C 总线数据信号
21	DIG5/KC5	数码管位选信号 5 / 键盘列信号 5
22	DIG4/KC4	数码管位选信号 4 / 键盘列信号 4
23	SA/KR0	数码管 a 段 / 键盘行信号 0
24	SB/KR1	数码管 b 段 / 键盘行信号 1

第3章 典型应用电路图

3.1 电路原理图

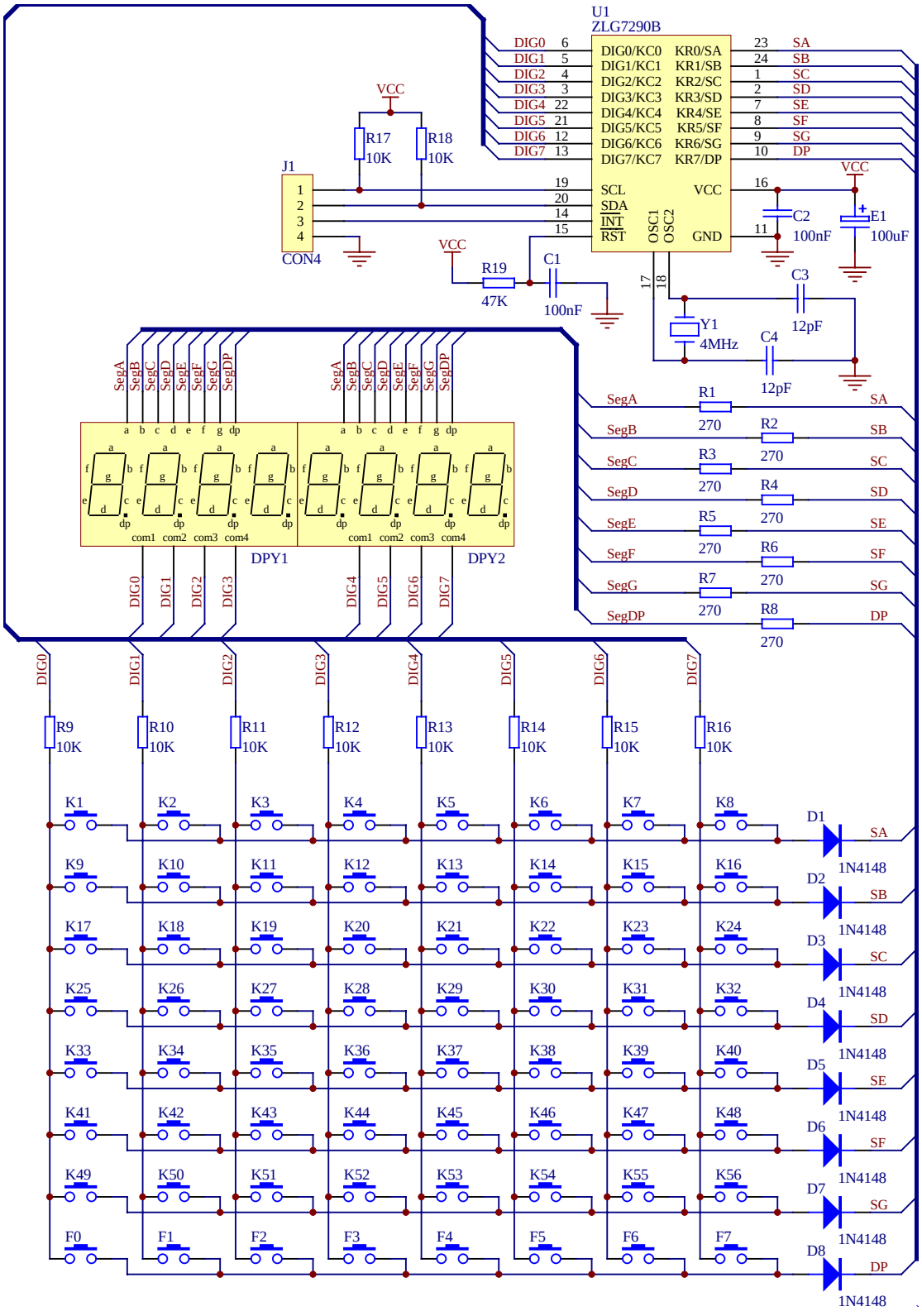

图 3.1 ZLG7290B 典型应用电路原理图

3.2 电路简析

在图 3.1 中，U1 就是 ZLG7290B。为了使电源更加稳定，一般要在 Vcc 到 GND 之间接入 47～470uF 的电解电容 E1。J1 是 ZLG7290B 与微控制器的接口，按照 I^2C 总线协议的要求，信号线 SCL 和 SDA 上必须要分别加上上拉电阻，其典型值是 10KΩ。晶振 Y1 通常取值 4MHz，调节电容 C3 和 C4 通常取值在 10pF 左右。复位信号是低电平有效，一般只需外接简单的 RC 复位电路，也可以通过直接拉低 $\overline{\text{RST}}$ 引脚的方法进行复位。

数码管必须是共阴式的，不能直接使用共阳式的。DPY1 和 DPY2 是 4 位联体式数码管，共同组成完整的 8 位。当然还可以采用其它的组合方式，如 4 只双联体式数码管。数码管在工作时要消耗较大的电流，R1～R8 是限流电阻，典型值是 270Ω。如果要增大数码管的亮度，可以适当减小电阻值，最低 200Ω。

64 只按键中，前 56 个按键是普通按键 K1～K56，最后 8 个为功能键 F0～F7。键盘电阻 R9～R16 的典型值是 10KΩ。数码管扫描线和键盘扫描线是共用的，所以二极管 D1～D8 是必须的，有了它们就可以防止按键干扰数码管显示的情况发生。在多数应用当中可能不需要太多的按键，这时可以按行或按列裁减键盘。裁减后相应行的二极管或相应列的电阻可以省略。如果完全不使用数码管，则原来用到的所有限流电阻 R1～R8 也都可以省略，这时 ZLG7290B 消耗的电流大大降低，典型值为 1mA。

第4章 功能详解

4.1 功能概述

如图 3.1 所示，ZLG7290B 可以扫描管理多达 64 个按键，K1～K56 为普通按键，F0～F7 为功能键。普通按键还有连击检测功能。

ZLG7290B 内部有 8 个显示缓冲寄存器 DpRam0～DpRam7，它们直接决定数码管显示的内容。ZLG7290B 提供有两种显示控制方式，一种是直接向显存写入字型数据，另一种是通过向命令缓冲寄存器写入控制指令实现自动译码显示。

访问这些寄存器需要通过 I^2C 总线接口来实现。ZLG7290B 的 I^2C 总线器件地址是 70H（写操作）和 71H（读操作）。访问内部寄存器要通过“子地址”来实现。有关 I^2C 总线协议的介绍请参见第 7 章；第 8 章还给出了 I^2C 总线软件包，比较完整和规范；第 10 章中的程序详细地演示了内部寄存器和控制命令的用法。

4.2 寄存器详解

4.2.1 系统寄存器 SystemReg（地址：00H）

系统寄存器的第 0 位（LSB）称作 KeyAvi，标志着按键是否有效，0－没有按键被按下，1－有某个按键被按下。SystemReg 寄存器的其它位暂时没有定义。当按下某个键时，ZLG7290B 的 $\overline{INT}$ 引脚会产生一个低电平的中断请求信号。当读走键值后，中断信号就会自动撤销。而 KeyAvi 也同时予以反映。正常情况下，微控制器只需要判断 $\overline{INT}$ 引脚就可以了。通过不断查询 KeyAvi 位也能判断是否有键按下，这样就可以节省微控制器的一根 I/O 口线，但是代价是 I^2C 总线处于频繁的活动状态，多消耗电流并且不利于抗干扰。

4.2.2 键值寄存器 Key（地址：01H）

如果某个普通键（图 3.1 中的 K1～K56）被按下，则微控制器可以从键值寄存器 Key 中读取相应的键值 1～56。如果微控制器发现 ZLG7290B 的 $\overline{INT}$ 引脚产生了中断请求，而从 Key 中读到的键值是 0，则表示按下的可能是功能键。键值寄存器 Key 的值在被读走后自动变成 0。

4.2.3 连击计数器 RepeatCnt（地址：02H）

ZLG7290B 为普通键（图 3.1 中的 K1～K56）提供了连击计数功能。所谓连击是指按住某个普通键不松手，经过一两秒钟的延迟后（在 4MHz 下约为 2 秒），开始连续有效，连续有效间隔时间（在 4MHz 下约为 170 毫秒）在几十到几百个毫秒。这一特性跟电脑上的键盘很类似。在微控制器能够及时响应按键中断并及时读取键值的前提下，当按住某个普通键一直不松手时：首先会产生一次中断信号，这时连击计数器 RepeatCnt 的值仍然是 0；经过一两秒延迟后，会连续产生中断信号，每中断一次 RepeatCnt 就自动加 1；当 RepeatCnt 计数到 255 时就不再增加，而中断信号继续有效。在此期间，键值寄存器的值每次都会产生。

4.2.4 功能键寄存器 FunctionKey（地址：03H）

ZLG7290B 还提供有 8 个功能键（图 3.1 中的 F0～F7）。功能键常常是配合普通键一起使用的，就像电脑键盘上的 Shift、Ctrl 和 Alt 键。当然功能键也可以单独去使用，就像电脑键盘上的 F1～F12。当按下某个功能键时，在 $\overline{INT}$ 引脚也会像按普通键那样产生中断信号。功能键的键值是被保存在 FunctionKey 寄存器中的。功能键寄存器 FunctionKey 的初始值是 FFH，每一个位对应一个功能键，第 0 位（LSB）对应 F0，第 1 位对应 F1，依次类推，第 7 位（MSB）对应 F7。某一功能键被按下时，相应的 FunctionKey 位就清零。功能键还有一

个特性就是“二次中断”，按下时产生一次中断信号，抬起时又会产生一次中断信号；而普通键只会在被按下时产生一次中断。

4.2.5 命令缓冲区 CmdBuf0 和 CmdBuf1（地址：07H 和 08H）

通过向命令缓冲区写入相关的控制命令可以实现段寻址、下载显示数据、控制闪烁等功能。详见第 4.3 节。

4.2.6 闪烁控制寄存器 FlashOnOff（地址：0CH）

FlashOnOff 寄存器决定闪烁频率和占空比。复位值为 0111,0111B。高 4 位表示闪烁时亮的持续时间，低 4 位表示闪烁时灭的持续时间。改变 FlashOnOff 的值，可以同时改变闪烁频率和占空比。FlashOnOff 取值 00H 时可获得最快的闪烁速度，在 4MHz 下，亮或灭的持续时间最小单位约为 280ms。特别说明：单独设置 FlashOnOff 寄存器的值，并不会看到显示闪烁，而应该配合闪烁控制命令（详见第 4.3 节）一起使用。

4.2.7 扫描位数寄存器 ScanNum（地址：0DH）

ScanNum 寄存器决定扫描显示的位数，取值 0～7，对应 1～8 位。复位值是 7，即数码管的 8 个位都扫描显示。实际应用中可能需要显示的位数不足 8 位，例如只显示 3 位，这时可以把 ScanNum 的值设置为 2，则数码管的第 0、1、2 位被扫描显示，而第 3～7 位不会被分配扫描时间，所以不显示。数码管的扫描位数减少后，有用的显示位由于分配的扫描时间更多因而显示亮度得以提高。ScanNum 寄存器的值为 0 时，只有数码管的第 0 位在显示，亮度达到最大。

4.2.8 显示缓冲区 DpRam0～DpRam7（地址：10H～17H）

DpRam0～DpRam7 这 8 个寄存器的取值直接决定了数码管的显示内容。每个寄存器的 8 个位分别对应数码管的 a,b,c,d,e,f,dp 段，MSB 对应 a，LSB 对应 dp。例如大写字母 H 的字型数据为 6EH（不带小数点）或 6FH（带小数点）。

4.3 控制命令详解

寄存器 CmdBuf0（地址：07H）和 CmdBuf1（地址：08H）共同组成命令缓冲区。通过向命令缓冲区写入相关的控制命令可以实现段寻址、下载显示数据、控制闪烁等功能。

4.3.1 段寻址（SegOnOff）

D7	D6	D5	D4	D3	D2	D1	D0		D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	1		on	0	S_5	S_4	S_3	S_2	S_1	S_0

在段寻址命令中，8 位数码管被看成是 64 个段，每一个段实际上就是一只独立的 LED。第 1 字节 0000,0001B 是命令字；on 表示该段是否点亮，0－灭，1－亮；$S_5S_4S_3S_2S_1S_0$ 是 6 位段地址，取值 0～63。在某 1 位数码管内，各段的亮或灭的顺序按照 a,b,c,d,e,f,g,dp 进行。

4.3.2 下载数据并译码（Download）

D7	D6	D5	D4	D3	D2	D1	D0		D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	0	A_3	A_2	A_1	A_0		dp	flash	0	d_4	d_3	d_2	d_1	d_0

在指令格式中，高 4 位的 0110 是命令字段；A3A2A1A0 是数码管显示数据的位地址（其中 A3 留作以后扩展之用，实际使用时取 0 即可），位地址编号按从左到右的顺序依次为 0,1,2,3,4,5,6,7（以第 3 章中的图 3.1 为准）；dp 控制小数点是否点亮，0－点亮，1－熄灭；flash 表示是否要闪烁，0－正常显示，1－闪烁；$d_4d_3d_2d_1d_0$ 是要显示的数据，包括 10 种数字和 21 种字母。显示数据按照表 4.1 中的规则进行译码：

表 4.1 下载数据并译码命令的数据表

$d_4d_3d_2d_1d_0$（二进制）					$d_4d_3d_2d_1d_0$（十六进制）	显示结果
0	0	0	0	0	00H	0
0	0	0	0	1	01H	1
0	0	0	1	0	02H	2
0	0	0	1	1	03H	3
0	0	1	0	0	04H	4
0	0	1	0	1	05H	5
0	0	1	1	0	06H	6
0	0	1	1	1	07H	7
0	1	0	0	0	08H	8
0	1	0	0	1	09H	9
0	1	0	1	0	0AH	A
0	1	0	1	1	0BH	b
0	1	1	0	0	0CH	C
0	1	1	0	1	0DH	d
0	1	1	1	0	0EH	E
0	1	1	1	1	0FH	F
1	0	0	0	0	10H	G
1	0	0	0	1	11H	H
1	0	0	1	0	12H	i
1	0	0	1	1	13H	J
1	0	1	0	0	14H	L
1	0	1	0	1	15H	o
1	0	1	1	0	16H	p
1	0	1	1	1	17H	q
1	1	0	0	0	18H	r
1	1	0	0	1	19H	t
1	1	0	1	0	1AH	U
1	1	0	1	1	1BH	y
1	1	1	0	0	1CH	c
1	1	1	0	1	1DH	h
1	1	1	1	0	1EH	T
1	1	1	1	1	1FH	（无显示）

4.3.3 闪烁控制（Flash）

D7	D6	D5	D4	D3	D2	D1	D0		D7	D6	D5	D4	D3	D2	D1	D0
0	1	1	1	x	x	x	x		F_7	F_6	F_5	F_4	F_3	F_2	F_1	F_0

在命令格式中，高 4 位的 0111 是命令字段；xxxx 表示无关位，通常取值 0000；第 2 字节的 Fn（n＝0～7）控制数码管相应位的闪烁属性，0－正常显示，1－闪烁。复位后，所有位都不闪烁。

第5章 实际应用中要注意的若干问题

5.1 ZLG7290B 安装布局注意事项

ZLG7290B 可广泛应用于仪器仪表，工业控制器，条形显示器，控制面板等领域。在实际应用中，控制面板和主机板往往是分离的，它们之间有几十厘米的距离，要用长长的排线相连。键盘和数码管一般都位于控制面板上，主控制器则在主机板上。在设计时千万注意：ZLG7290B 一定要跟着控制面板走，而不要放在主机板上。ZLG7290B 驱动数码管显示采用的是动态扫描法，为了防止显示出现闪烁，采用了比较高的扫描频率。扫描键盘同样用的也是频率较高的信号。如果 ZLG7290B 放在主机板上，这些扫描信号势必要走长线，而高频信号最忌讳走长线了，这容易导致显示混乱、按键失灵等故障。如果 ZLG7290B 放在控制面板上，由于走的是短线，就不易出现上述问题了。不必担心 ZLG7290B 与主控制器之间通信的 I^2C 总线会有问题。因为 I^2C 总线的通信速率是由主控制器控制的，可以做得低一些，所以允许走长线。

5.2 复位引脚可以由主控制器直接控制

在工业控制应用中，为了增强抗干扰能力，建议采用独立的稳定直流电源给 ZLG7290B 供电，Vcc 与 GND 之间的电容也要相应加大。另外复位引脚最好由主控制器来控制，每隔几分钟强制复位一次，复位脉冲宽度可以在 20ms 左右，一闪而过，肉眼很难察觉。定时强制复位可以有效防止偶尔由于电磁干扰而产生的显示不正常和按键失灵的现象。

5.3 驱动 1 英寸以上的大数码管时，要另外加驱动电路

ZLG7290B 的驱动能力毕竟是有限的，如果直接驱动 1 英寸以上的大数码管则可能会导致显示亮度不够。这时可以适当减小限流电阻（最低 200Ω）以增加亮度。如果亮度仍然不够，就必须另外添加驱动电路。更深入的讨论请参见第 6.3 节。

5.4 降低晶振频率

在 ZLG7290B 的典型应用电路图当中，晶振用的是 4MHz。在一般情况下，能够稳定地工作。但是在电磁环境恶劣的现场，建议把晶振频率再降低一些，降到 1～3MHz。许多本来“有问题”的电路，在把晶振速度降下来之后就完全正常了。晶振频率降低后，I^2C 总线的通信速率也要适当降低。ZLG7290B 的闪烁显示功能将受到影响，闪烁速度将因晶振频率的下降而跟着变慢，这时要适当调整闪烁控制寄存器 FlashOnOff 的数值（详见第 4.2.6 小节）。

第6章 特殊应用

6.1 只使用键盘

ZLG7290B 在某些应用中，可能不需要使用数码管，而只使用其键盘扫描管理功能，这时，工作电流可降至 1mA。只使用键盘的具体用法请参考图 6.1。

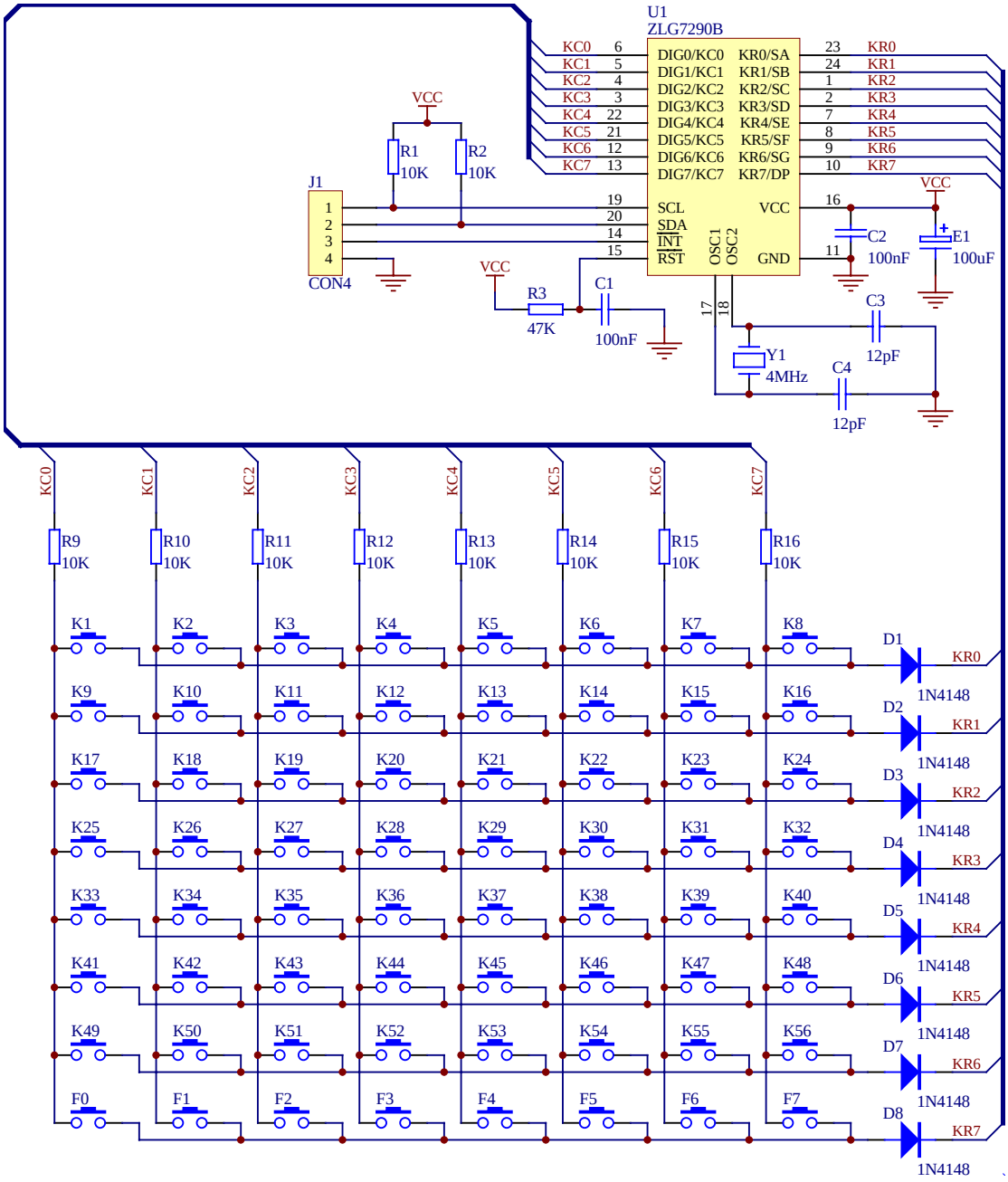

图 6.1 只使用键盘的应用电路图

6.2 只使用数码管

ZLG7290B 在某些应用中，可能不需要使用键盘，而只使用其数码管显示驱动功能。用于键盘扫描的电阻和二极管都可以去掉不要，电路因此大为简化。只使用数码管的具体用法请参考图 6.2。

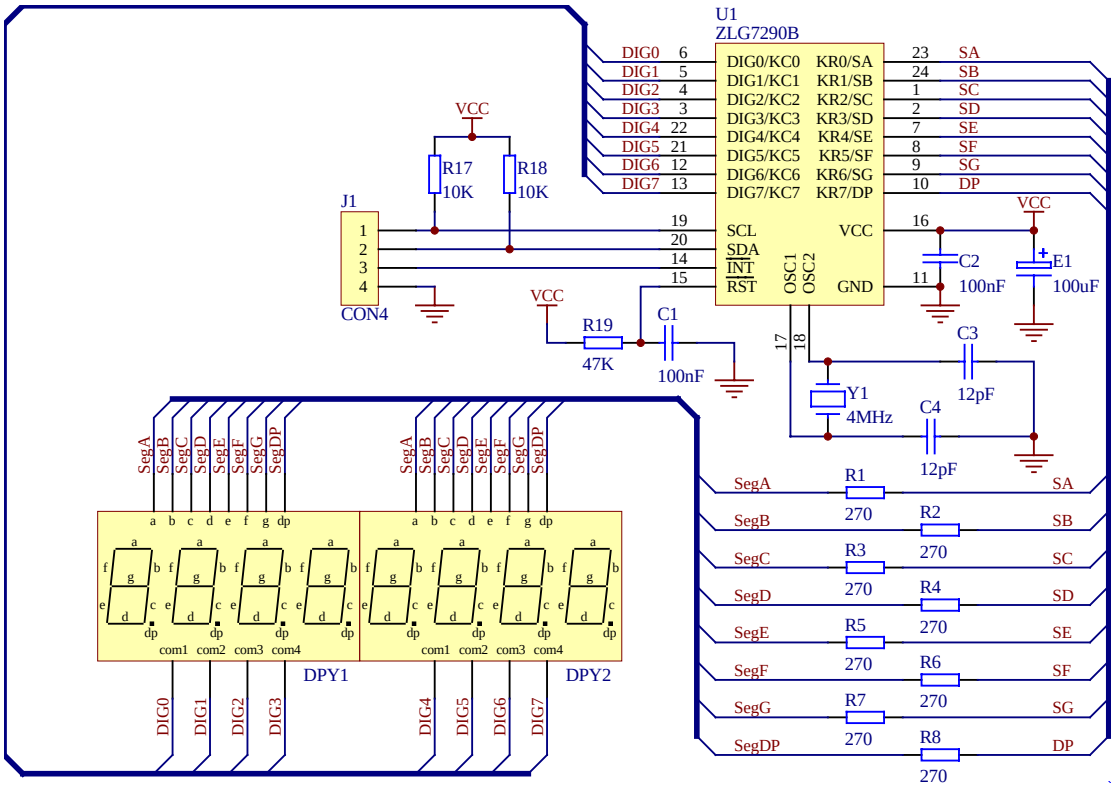

图 6.2 只使用数码管的应用电路图

6.3 驱动大型数码管的方法

ZLG7290B 的驱动能力毕竟有限，如果要使用 1 英寸以上的大型数码管，则显示亮度可能不够，这时可以考虑适当减小数码管的限流电阻（参见第 3 章图 3.1 中的 R1～R8）的阻值以增加亮度，阻值最小为 200Ω。如果亮度依然不够，就必须要另外加入功率驱动电路。采用功率晶体管作为驱动电路是很容易想到的方案，其原理如图 6.3 所示。

在图 6.3 中，以第 1 路驱动为例，ZLG7290B 的段选信号 SA 是高电平有效的，DIG0 则是低电平有效。SA 为高电平时，使 Q1（NPN 通用型）导通，Q1 的导通又使得 Q2（PNP 功率型）导通；DIG0 为低电平时，使 Q17（PNP 通用型）导通，Q17 的导通又使得 Q18（NPN 功率型）导通；这样就有电流从 Vs 经限流电阻 R3（功率电阻，阻值视具体情况而定）和数码管流到 GND，于是相应的数码管字段就被点亮。如果 SA 和 DIG0 信号不是一高一低的组合，则相应的数码管字段就不会亮。这种接法完全符合 ZLG7290B 的动态扫描工作方式，并且不会影响键盘扫描管理功能。注意，图中的电源 Vs 可以是与 ZLG7290B 相同的电源 Vcc，也可以是单独的高压电源（20V 以内）。

用分立功率晶体管作为驱动电路显然太麻烦，要动用 16 只通用型晶体管和 16 只功率型晶体管，以及若干只电阻。那么有没有替代它们的功率集成电路呢？Allegro 公司的 8 路达林顿阵列 UDN2981A 和 UDN2596A 就是一对很好的组合，它们的驱动电流高达 1A。其内部结构请参考图 6.4，其中 UDN2981A 相当于图 6.3 中 Q1 与 Q2 的组合，UDN2596A 相当于图 6.3 中 Q17 与 Q18 的组合。ZLG7290B 与它们相配合一起驱动大型数码管的完整电路图请参考图 6.5，可以看出，这比分立功率晶体管的电路简单多了。图 6.5 中 UDN2981A 的电源 Vs 可以是单独的高压电源（20V 以内），R1～R8 是限流电阻（功率型），阻值视具体情况而定。

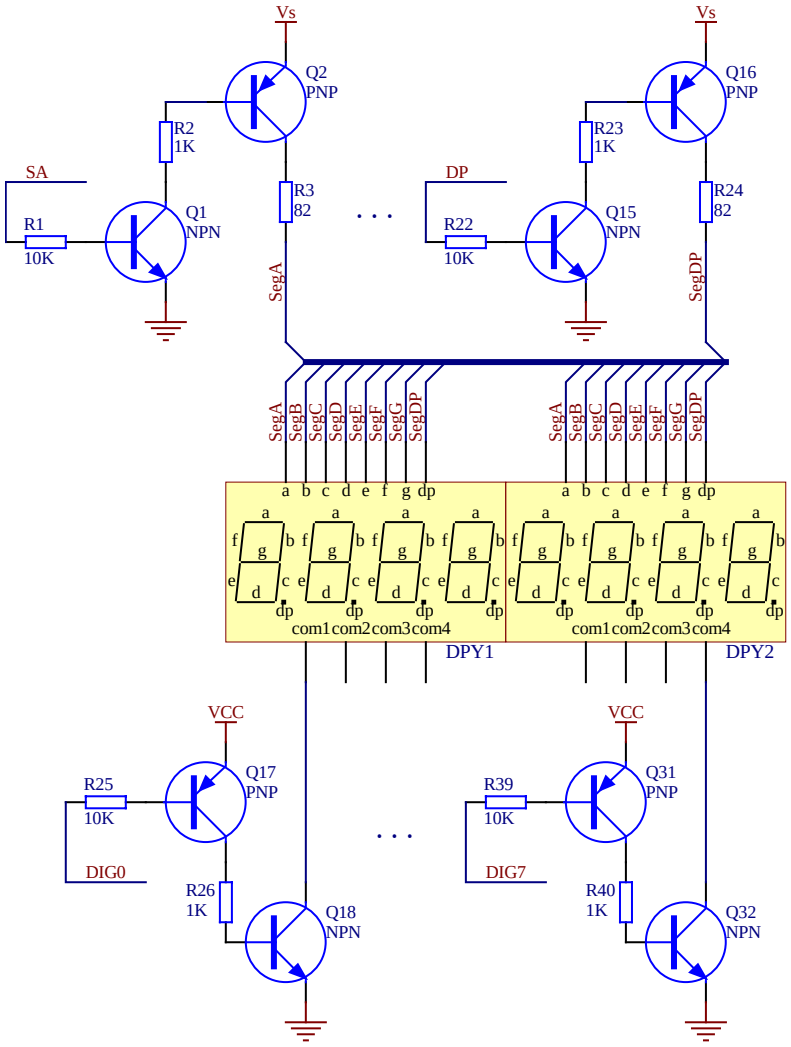

图 6.3 使用分立功率晶体管驱动大型数码管

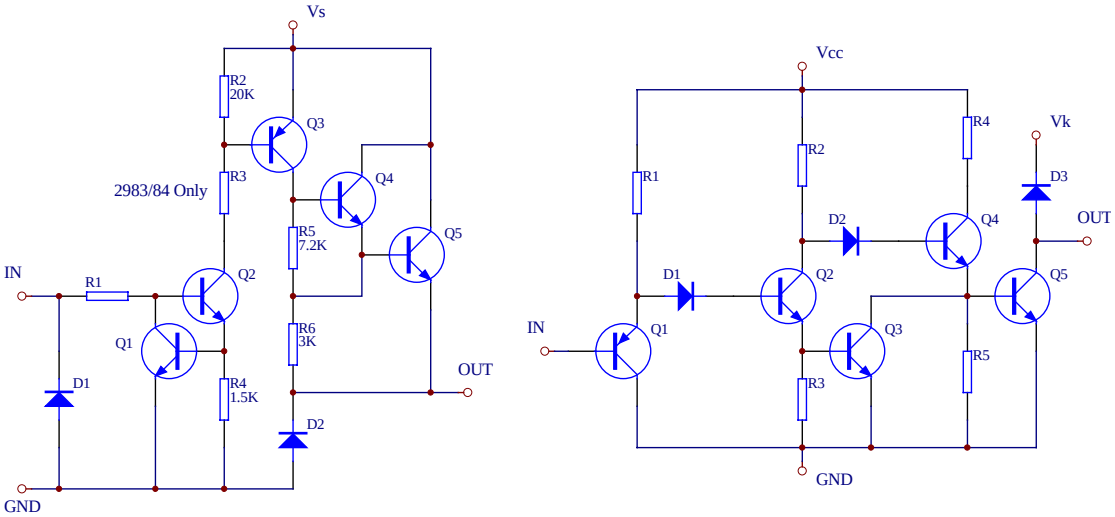

图 6.4 UDN2981A（左）和 UDN2596A（右）其中一路的内部结构

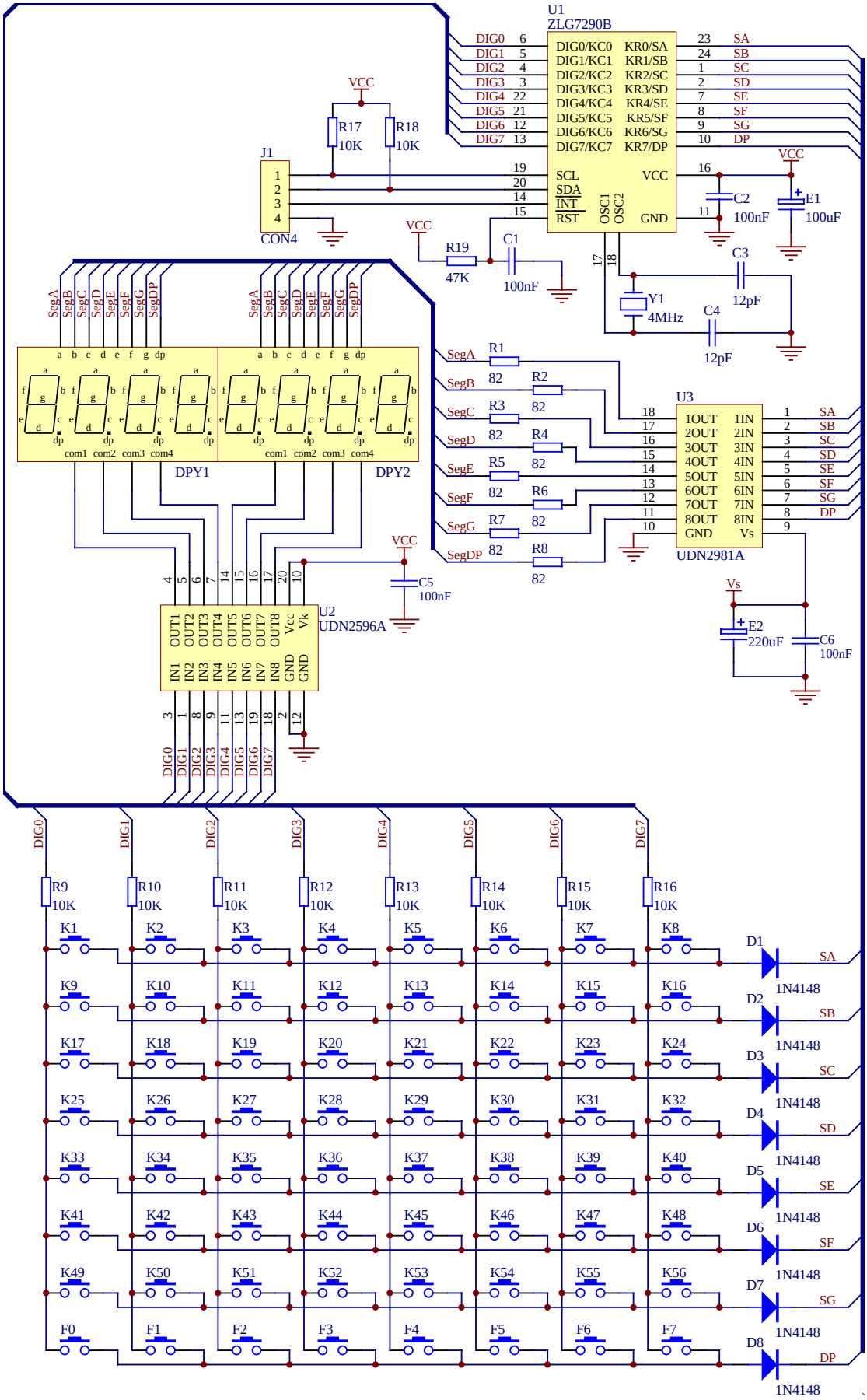

图 6.5 利用达林顿阵列驱动大型数码管

第7章 I²C 总线简介

ZLG7290B 与微控制器的接口形式是 I²C 串行总线，因此有必要简要地介绍一下 I²C 总线协议标准。限于篇幅，这里不可能列出协议的全部内容，如果您希望了解更深层次的问题，请参考飞利浦公司的网站（http://www.semiconductors.philips.com/），或者周立功单片机公司的网站（http://www.zlgmcu.com）上相关的文档，或者是何立民教授编著的《I²C 总线应用系统设计》。第 8 章还给出了用 C51 编写的 I²C 总线软件包，以供参考学习。

7.1 I²C 总线概述

飞利浦（Philips）于 20 多年前发明了一种简单的双向二线制串行通信总线，这个总线被称为 Inter-IC 或者 I²C 总线。目前 I²C 总线已经成为业界嵌入式应用的标准解决方案，被广泛地应用在各式各样基于微控器的专业、消费与电信产品中，作为控制、诊断与电源管理总线。多个符合 I²C 总线标准的器件都可以通过同一条 I²C 总线进行通信，而不需要额外的地址译码器。由于 I²C 是一种两线式串行总线，因此简单的操作特性成为它快速崛起成为业界标准的关键因素。

7.2 I²C 总线的信号线

I²C 总线只需要由两根信号线组成，一根是串行数据线 SDA，另一根是串行时钟线 SCL。

一般具有 I²C 总线的器件其 SDA 和 SCL 引脚都是漏极开路（或集电极开路）输出结构。因此实际使用时，SDA 和 SCL 信号线都必须要加上拉电阻（Rp，Pull-Up Resistor）。上拉电阻一般取值 3～10KΩ。开漏结构的好处是：当总线空闲时，这两条信号线都保持高电平，几乎不消耗电流；电气兼容性好，上拉电阻接 5V 电源就能与 5V 逻辑器件接口，上拉电阻接 3V 电源又能与 3V 逻辑器件接口；因为是开漏结构，所以不同器件的 SDA 与 SDA 之间、SCL 与 SCL 之间可以直接相连，不需要额外的转换电路。

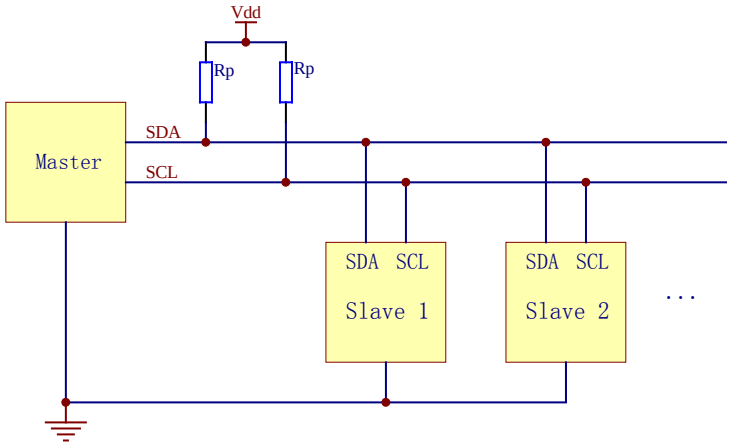

图 7.1 I²C 总线信号连接示意图

7.3 I²C 总线基本概念

- 发送器（Transmitter）：发送数据到总线的器件；
- 接收器（Receiver）：从总线接收数据的器件；
- 主机（Master）：初始化发送、产生时钟信号和终止发送的器件；
- 从机（Slave）：被主机寻址的器件。

I²C 总线是双向传输的总线，因此主机和从机都可能成为发送器和接收器。如果主机向从机发送数据，则主机是发送器，而从机是接收器；如果主机从从机读取数据，则主机是接

收器，而从机是发送器。

7.4 I²C 总线数据传送速率

I²C 总线的通信速率受主机控制，能快能慢。但是最高速率是有限制的，I²C 总线上数据的传输速率在标准模式（Standard-mode）下最快可达 100Kb/s。

7.5 I²C 总线上数据的有效性（Data validity）

数据线 SDA 的电平状态必须在时钟线 SCL 处于高电平期间保持稳定不变。SDA 的电平状态只有在 SCL 处于低电平期间才允许改变。但是在 I²C 总线的起始和结束时例外。

某些其它的串行总线协议可能规定数据在时钟信号的边沿（上升沿或下降沿）有效，而 I²C 总线则是电平有效。

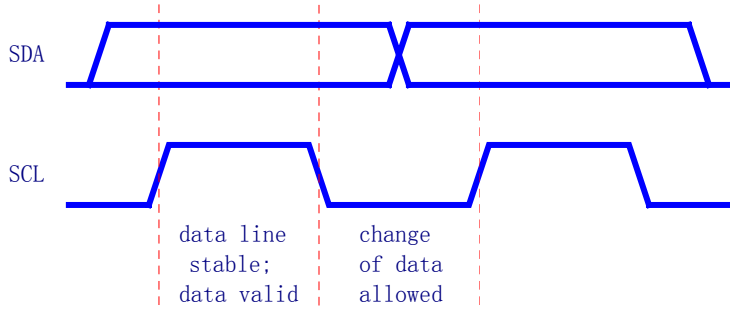

图 7.2 I²C 总线上数据有效性的示意图

7.6 起始条件和停止条件（START and STOP conditions）

起始条件：当 SCL 处于高电平期间时，SDA 从高电平向低电平跳变时产生起始条件。总线在起始条件产生后便处于忙的状态。起始条件常常简记为 S。

停止条件：当 SCL 处于高电平期间时，SDA 从低电平向高电平跳变时产生停止条件。总线在停止条件产生后处于空闲状态。停止条件简记为 P。

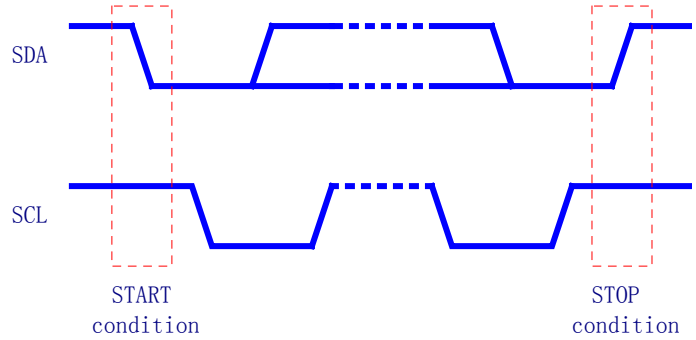

图 7.3 I²C 起始条件和停止条件示意图

7.7 从机地址（Slave Address）

I²C 总线不需要额外的地址译码器和片选信号。多个具有 I²C 总线接口的器件都可以连接到同一条 I²C 总线上，它们之间通过器件地址来区分。主机是主控器件，它不需要器件地址，其它器件都属于从机，要有器件地址。必须保证同一条 I²C 总线上所有从机的地址都是唯一确定的，不能有重复，否则 I²C 总线将不能正常工作。一般从机地址由 7 位地址位和一位读写标志（R/$\overline{W}$）组成，7 位地址占据高 7 位，读写位在最后。读写位是 0，表示主机将要向从机写入数据；读写位是 1，则表示主机将要从从机读取数据。

7.8 数据传输的基本格式

I^2C 总线以字节为单位收发数据。传输到 SDA 线上的每个字节必须为 8 位。每次传输的字节数量不受限制。首先传输的是数据的最高位（MSB，第 7 位），最后传输的是最低位（LSB，第 0 位）。另外，每个字节之后还要跟一个响应位，称为应答。

7.9 应答（Acknowledge）

在 I^2C 总线传输数据过程中，每传输一个字节，都要跟一个应答状态位。接收器接收数据的情况可以通过应答位来告知发送器。应答位的时钟脉冲仍由主机产生，而应答位的数据状态则遵循“谁接收谁产生”的原则，即总是由接收器产生应答位。主机向从机发送数据时，应答位由从机产生；主机从从机接收数据时，应答位由主机产生。I^2C 总线标准规定：应答位为 0 表示接收器应答（ACK），常常简记为 A；为 1 则表示非应答（NACK），常常简记为 $\overline{A}$。发送器发送完 LSB 之后，应当释放 SDA 线（拉高 SDA，输出晶体管截止），以等待接收器产生应答位。

如果接收器在接收完最后一个字节的数据，或者不能再接收更多的数据时，应当产生非应答来通知发送器。发送器如果发现接收器产生了非应答状态，则应当终止发送。

7.10 基本的数据传输格式示意图

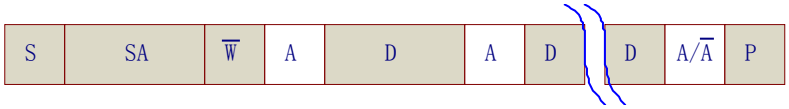

图 7.4 主机向从机发送数据的基本格式

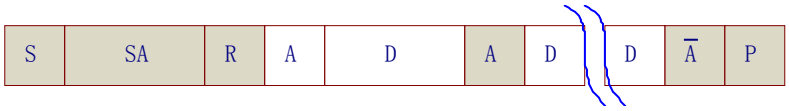

图 7.5 主机从从机接收数据的基本格式

在图 7.4 和图 7.5 中，各种符号的意义为：

- S：起始位（START）；
- SA：从机地址（Slave Address），7 位从机地址；
- $\overline{W}$：写标志位（Write），1 位写标志；
- R：读标志位（Read），1 位读标志；
- A：应答位（Acknowledge），1 位应答；
- $\overline{A}$：非应答位（Not Acknowledge），1 位非应答；
- D：数据（Data），每个数据都必须是 8 位；
- P：停止位（STOP）；
- 阴影：主机产生的信号；
- 无阴影：从机产生的信号。

应当注意的是，与图 7.5 中的情况不同的是，在图 7.4 中，主机向从机发送最后一个字节的数据时，从机可能应答也可能非应答，但不管怎样主机都可以产生停止条件。如果主机在向从机发送数据（甚至包括从机地址在内）时检测到从机非应答，则应当及时停止传输。

7.11 传输一个字节数据的时序图

为了更清楚地了解 I^2C 总线的基本数据传输过程，下面画出了只传输 1 个字节的时序图，这是最基本的传输方式。在图 7.6 和图 7.7 中，SDA 信号线被画成了两个，一个是主机产生

的，另一个是从机产生的。实际上主机和从机的 SDA 信号线总是连接在一起的，是同一根 SDA。画成两个 SDA 有助于进一步理解在 I^2C 总线上主机和从机的不同行为。

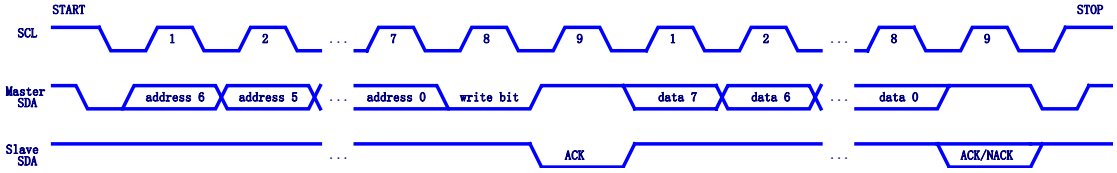

图 7.6 主机向从机发送 1 个字节数据的时序图

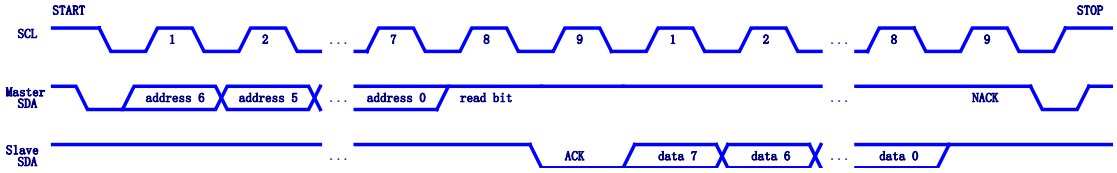

图 7.7 主机从从机接收 1 个字节数据的时序图

（注：图中文字较小，可放大后查看，下同。）

7.12 传输多个字节数据的时序图

主机连续向从机发送或从从机接收多个字节数据的情况也很容易理解，下面直接给出相关时序图。

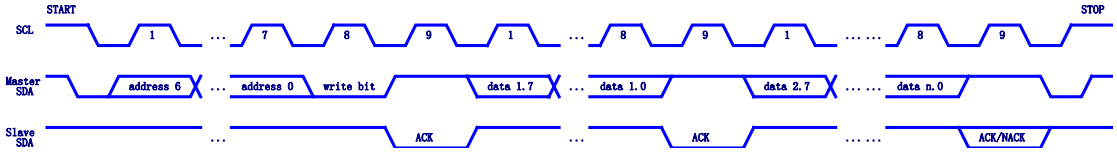

图 7.8 主机向从机连续发送多个字节数据的时序图

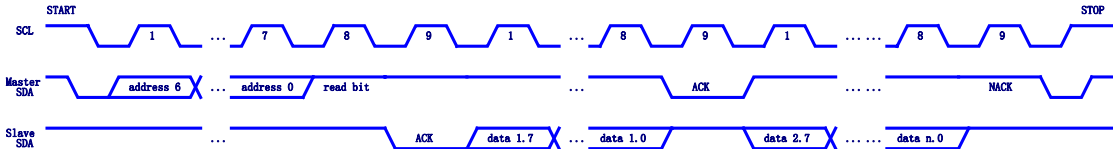

图 7.9 主机从从机连续接收多个字节数据的时序图

7.13 重复起始条件（Repeated START condition）

主机与从机进行通信时，有时需要切换数据的收发方向，例如访问某一具有 I^2C 总线接口的 E^2PROM 存储器时，主机先向存储器输入存储单元的地址信息（发送数据），然后再读取其中的存储内容（接收数据）。在切换数据的传输方向时，可以不必先产生停止条件再开始下次传输，而是直接再一次产生开始条件。I^2C 总线在已经处于忙的状态下，再一次直接产生起始条件的情况被称为重复起始条件。重复起始条件常常简记为 Sr。正常的起始条件和重复起始条件在物理波形上并没有什么不同，区别仅仅是在逻辑方面。在进行多字节数据传输过程中，只要数据的收发方向发生了切换，就要用到重复起始条件。

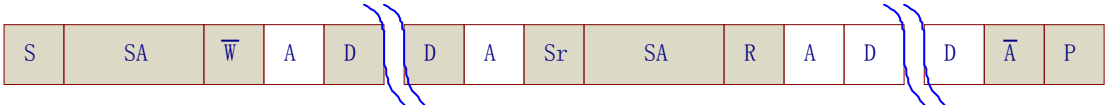

图 7.10 带有重复起始条件的多字节数据传输格式示意图

图 7.10 给出了带有重复起始条件的多字节数据传输格式示意图，图中的各种符号的意义与第 7.10 节中的相同。要特别注意图中重复起始条件 Sr 的用法。如果读者有兴趣的话，可以自行画出其对应的时序图。

7.14 无子地址器件与有子地址器件

带有 I^2C 总线的器件除了有从机地址（Slave Address）外，还可能有子地址（Sub-Address）。从机地址是指该器件在 I^2C 总线上被主机寻址的地址，而子地址是指该器件内部不同部件或存储单元的编址。例如，带 I^2C 总线接口的 E^2PROM 就是拥有子地址器件的典型代表。另外一些器件（只占少数）内部结构比较简单，可能没有子地址，只有必须的从机地址。与从机地址一样，子地址实际上也是像普通数据那样进行传输的，传输格式仍然是与数据相统一的，区分传输的到底是地址还是数据要靠收发双方具体的逻辑约定。子地址的长度必须由整数个字节组成，可能是单字节（8 位子地址），也可能是双字节（16 位子地址），还可能是 3 字节以上，这要看具体器件的规定。

在第 8 章的 I^2C 总线软件包中，已经同时考虑到了无子地址器件和有子地址器件的情况。

第8章 I²C 总线的 C51 驱动程序软件包

8.1 软件包说明

I²C 总线的 80C51 单片机 C51 驱动程序软件包由两个文件组成：“I2C.h”和“I2C.c”。头文件“I2C.h”包括了信号线 SDA 和 SCL 的 I/O 接口定义和用户函数的声明，C 语言文件“I2C.c”是这些函数的具体实现。这是一个用 C51 模拟 I²C 总线协议的精简版本，只考虑一主多从模式，不考虑多主模式，也不考虑时钟同步等问题。要想更清楚地了解本程序的细节，请参考 Philips 公司的相关协议标准。

8.2 使用方法

- 把文件“I2C.h”和“I2C.c”一起复制到您的工程文件夹下；
- 根据实际电路，修改 I²C 总线两根信号线 SCL 和 SDA 的定义；
- 通过宏定义 I2C_DELAY_VALUE 调整 I²C 总线的速度使其符合实际需要；
- 把文件“I2C.c”添加进工程中，在需要的地方包含头文件“I2C.h”；
- 在 main()函数的开始处，应当调用一次初始化函数 I2C_Init()；
- I2C_Puts()和 I2C_Gets()是 I²C 总线综合读写函数，请看清注释后再使用；
- 所有全局性质的标识符都以“I2C_”开头，可有效避免命名冲突问题；
- 注意：从机地址采用 7 位纯地址表示，不含读写位，即第 0～6 位是地址，第 7 位无效。这可能与其它程序的规定不同。

8.3 头文件程序清单

程序清单 8.1 模拟 I²C 总线的 C51 驱动程序头文件

```
/*
    I2C.h
    标准 80C51 单片机模拟 I2C 总线的主机程序头文件
    Copyright (c) 2005，广州周立功单片机发展有限公司
    All rights reserved.
    本程序仅供学习参考，不提供任何可靠性方面的担保；请勿用于商业目的
*/

#ifndef _I2C_H_
#define _I2C_H_

#include <reg52.h>

//模拟 I2C 总线的引脚定义
sbit I2C_SCL = P1^6;
sbit I2C_SDA = P1^7;

//定义 I2C 总线时钟的延时值，要根据实际情况修改，取值 1～255
//SCL 信号周期约为(I2C_DELAY_VALUE*4+15)个机器周期
#define I2C_DELAY_VALUE          5
```

```
//定义 I2C 总线停止后在下一次开始之前的等待时间，取值 1～65535
//等待时间约为(I2C_STOP_WAIT_VALUE*8)个机器周期
//对于多数器件取值为 1 即可；但对于某些器件来说，较长的延时是必须的
#define I2C_STOP_WAIT_VALUE      1

//I2C 总线初始化，使总线处于空闲状态
extern void I2C_Init();

//I2C 总线综合发送函数，向从机发送多个字节的数据
extern bit I2C_Puts
(
        unsigned char SlaveAddr,
        unsigned int SubAddr,
        unsigned char SubMod,
        char *dat,
        unsigned int Size
);

//I2C 总线综合接收函数，从从机接收多个字节的数据
extern bit I2C_Gets
(
        unsigned char SlaveAddr,
        unsigned int SubAddr,
        unsigned char SubMod,
        char *dat,
        unsigned int Size
);

#endif      //_I2C_H_
```

8.4 C 文件程序清单

程序清单 8.2 模拟 I^2C 总线的 C51 驱动程序 C 文件

```
/*
        I2C.c
        标准 80C51 单片机模拟 I2C 总线的主机程序
        Copyright (c) 2005，广州周立功单片机发展有限公司
        All rights reserved.
        本程序仅供学习参考，不提供任何可靠性方面的担保；请勿用于商业目的
*/

#include "I2C.h"

//定义延时变量，用于宏 I2C_Delay()
```

```
unsigned char data I2C_Delay_t;

/*
宏定义：I2C_Delay()
功能：延时，模拟 I2C 总线专用
*/
#define I2C_Delay()\
{\
    I2C_Delay_t = (I2C_DELAY_VALUE);\
    while ( --I2C_Delay_t != 0 );\
}

/*
函数：I2C_Init()
功能：I2C 总线初始化，使总线处于空闲状态
说明：在 main()函数的开始处，通常应当要执行一次本函数
*/
void I2C_Init()
{
    I2C_SCL = 1;
    I2C_Delay();
    I2C_SDA = 1;
    I2C_Delay();
}

/*
函数：I2C_Start()
功能：产生 I2C 总线的起始状态
说明：
    SCL 处于高电平期间，当 SDA 出现下降沿时启动 I2C 总线
    不论 SDA 和 SCL 处于什么电平状态，本函数总能正确产生起始状态
    本函数也可以用来产生重复起始状态
    本函数执行后，I2C 总线处于忙状态
*/
void I2C_Start()
{
    I2C_SDA = 1;
    I2C_Delay();
    I2C_SCL = 1;
    I2C_Delay();
    I2C_SDA = 0;
    I2C_Delay();
    I2C_SCL = 0;
    I2C_Delay();
```

```
}

/*
函数：I2C_Write()
功能：向 I2C 总线写 1 个字节的数据
参数：
    dat：要写到总线上的数据
*/
void I2C_Write(char dat)
{
    unsigned char t = 8;
    do
    {
        I2C_SDA = (bit)(dat & 0x80);
        dat <<= 1;
        I2C_SCL = 1;
        I2C_Delay();
        I2C_SCL = 0;
        I2C_Delay();
    } while ( --t != 0 );
}

/*
函数：I2C_Read()
功能：从从机读取 1 个字节的数据
返回：读取的一个字节数据
*/
char I2C_Read()
{
    char dat;
    unsigned char t = 8;
    I2C_SDA = 1;    //在读取数据之前，要把 SDA 拉高
    do
    {
        I2C_SCL = 1;
        I2C_Delay();
        dat <<= 1;
        if ( I2C_SDA ) dat |= 0x01;
        I2C_SCL = 0;
        I2C_Delay();
    } while ( --t != 0 );
    return dat;
}
```

```
/*
函数：I2C_GetAck()
功能：读取从机应答位
返回：
      0：从机应答
      1：从机非应答
说明：
      从机在收到每个字节的数据后，要产生应答位
      从机在收到最后 1 个字节的数据后，一般要产生非应答位
*/
bit I2C_GetAck()
{
      bit ack;
      I2C_SDA = 1;
      I2C_Delay();
      I2C_SCL = 1;
      I2C_Delay();
      ack = I2C_SDA;
      I2C_SCL = 0;
      I2C_Delay();
      return ack;
}

/*
函数：I2C_PutAck()
功能：主机产生应答位或非应答位
参数：
      ack=0：主机产生应答位
      ack=1：主机产生非应答位
说明：
      主机在接收完每一个字节的数据后，都应当产生应答位
      主机在接收完最后一个字节的数据后，应当产生非应答位
*/
void I2C_PutAck(bit ack)
{
      I2C_SDA = ack;
      I2C_Delay();
      I2C_SCL = 1;
      I2C_Delay();
      I2C_SCL = 0;
      I2C_Delay();
}

/*
```

```
函数：I2C_Stop()
功能：产生 I2C 总线的停止状态
说明：
    SCL 处于高电平期间，当 SDA 出现上升沿时停止 I2C 总线
    不论 SDA 和 SCL 处于什么电平状态，本函数总能正确产生停止状态
    本函数执行后，I2C 总线处于空闲状态
*/
void I2C_Stop()
{
    unsigned int t = I2C_STOP_WAIT_VALUE;
    I2C_SDA = 0;
    I2C_Delay();
    I2C_SCL = 1;
    I2C_Delay();
    I2C_SDA = 1;
    I2C_Delay();
    while ( --t != 0 );        //在下一次产生 Start 之前，要加一定的延时
}

/*
函数：I2C_Puts()
功能：I2C 总线综合发送函数，向从机发送多个字节的数据
参数：
    SlaveAddr：从机地址（7 位纯地址，不含读写位）
    SubAddr：从机的子地址
    SubMod：子地址模式，0－无子地址，1－单字节子地址，2－双字节子地址
    *dat：要发送的数据
    Size：数据的字节数
返回：
    0：发送成功
    1：在发送过程中出现异常
说明：
    本函数能够很好地适应所有常见的 I2C 器件，不论其是否有子地址
    当从机没有子地址时，参数 SubAddr 任意，而 SubMod 应当为 0
*/
bit I2C_Puts
(
    unsigned char SlaveAddr,
    unsigned int SubAddr,
    unsigned char SubMod,
    char *dat,
    unsigned int Size
)
{
```

```
//定义临时变量
    unsigned char i;
    char a[3];
//检查长度
    if ( Size == 0 ) return 0;
//准备从机地址
    a[0] = (SlaveAddr << 1);
//检查子地址模式
    if ( SubMod > 2 ) SubMod = 2;
//确定子地址
    switch ( SubMod )
    {
    case 0:
        break;
    case 1:
        a[1] = (char)(SubAddr);
        break;
    case 2:
        a[1] = (char)(SubAddr >> 8);
        a[2] = (char)(SubAddr);
        break;
    default:
        break;
    }
//发送从机地址，接着发送子地址（如果有子地址的话）
    SubMod++;
    I2C_Start();
    for ( i=0; i<SubMod; i++ )
    {
        I2C_Write(a[i]);
        if ( I2C_GetAck() )
        {
            I2C_Stop();
            return 1;
        }
    }
//发送数据
    do
    {
        I2C_Write(*dat++);
        if ( I2C_GetAck() ) break;
    } while ( --Size != 0 );
//发送完毕，停止 I2C 总线，并返回结果
    I2C_Stop();
```

```
    if ( Size == 0 )
    {
        return 0;
    }
    else
    {
        return 1;
    }
}

/*
函数：I2C_Gets()
功能：I2C 总线综合接收函数，从从机接收多个字节的数据
参数：
    SlaveAddr：从机地址（7 位纯地址，不含读写位）
    SubAddr：从机的子地址
    SubMod：子地址模式，0－无子地址，1－单字节子地址，2－双字节子地址
    *dat：保存接收到的数据
    Size：数据的字节数
返回：
    0：接收成功
    1：在接收过程中出现异常
说明：
    本函数能够很好地适应所有常见的 I2C 器件，不论其是否有子地址
    当从机没有子地址时，参数 SubAddr 任意，而 SubMod 应当为 0
*/
bit I2C_Gets
(
    unsigned char SlaveAddr,
    unsigned int SubAddr,
    unsigned char SubMod,
    char *dat,
    unsigned int Size
)
{
//定义临时变量
    unsigned char i;
    char a[3];
//检查长度
    if ( Size == 0 ) return 0;
//准备从机地址
    a[0] = (SlaveAddr << 1);
//检查子地址模式
    if ( SubMod > 2 ) SubMod = 2;
```

```
//如果是有子地址的从机，则要先发送从机地址和子地址
    if ( SubMod != 0 )
    {
    //确定子地址
        if ( SubMod == 1 )
        {
            a[1] = (char)(SubAddr);
        }
        else
        {
            a[1] = (char)(SubAddr >> 8);
            a[2] = (char)(SubAddr);
        }
    //发送从机地址，接着发送子地址
        SubMod++;
        I2C_Start();
        for ( i=0; i<SubMod; i++ )
        {
            I2C_Write(a[i]);
            if ( I2C_GetAck() )
            {
                I2C_Stop();
                return 1;
            }
        }
    }
//这里的 I2C_Start()对于有子地址的从机是重复起始状态
//对于无子地址的从机则是正常的起始状态
    I2C_Start();
//发送从机地址
    I2C_Write(a[0]+1);
    if ( I2C_GetAck() )
    {
        I2C_Stop();
        return 1;
    }
//接收数据
    for (;;)
    {
        *dat++ = I2C_Read();
        if ( --Size == 0 )
        {
            I2C_PutAck(1);
            break;
```

```
            }
            I2C_PutAck(0);
        }
//接收完毕，停止 I2C 总线，并返回结果
        I2C_Stop();
        return 0;
}
```

第9章 ZLG7290B 的 C51 驱动程序软件包

9.1 软件包说明

ZLG7290B 的 80C51 单片机 C51 驱动程序软件包由两个文件组成：“ZLG7290.h”和“ZLG7290.c”。头文件“ZLG7290.h”包括了中断信号 $\overline{\text{INT}}$ 的 I/O 接口定义、内部寄存器定义和用户函数的声明，C 语言文件“ZLG7290.c”则是这些函数的具体实现。在 ZLG7290B 的软件包中，还将用到第 8 章中的 I^2C 总线 C51 驱动程序软件包。

9.2 使用方法

- 把第 8 章中的文件“I2C.h”和“I2C.c”一起复制到您的工程文件夹下；
- 按照第 8.2 节的使用说明修改 I^2C 总线软件包的相关定义；
- 修改 I2C_DELAY_VALUE 的值，4MHz 下不小于 12；
- 修改 I2C_STOP_WAIT_VALUE 的值，4MHz 下不小于 120；
- 把文件“I2C.c”添加进工程中，在需要的地方包含头文件“I2C.h”；
- 把文件“ZLG7290.h”和“ZLG7290.c”一起复制到您的工程文件夹下；
- 修改“ZLG7290.h”中 $\overline{\text{INT}}$ 引脚的 I/O 定义，使其与实际电路一致；
- 把文件“ZLG7290.c”添加进工程中；
- 在 main()函数的开始处，应当调用一次初始化函数 I2C_Init()；
- 以后在程序中可以直接使用“ZLG7290.h”中声明的用户函数了。

9.3 头文件程序清单

程序清单 9.1 ZLG7290B 的 C51 驱动程序头文件

```
/*
    ZLG7290.h
    数码管显示与键盘管理芯片 ZLG7290 的标准 80C51 驱动程序头文件
    Copyright (c) 2005，广州周立功单片机发展有限公司
    All rights reserved.
    本程序仅供学习参考，不提供任何可靠性方面的担保；请勿用于商业目的
*/


#ifndef _ZLG7290_H_
#define _ZLG7290_H_

#include <reg52.h>

//ZLG7290 中断请求信号的引脚定义
sbit ZLG7290_pinINT = P3^2;

//定义 ZLG7290 在 I2C 总线协议中的从机地址
//这是 7 位纯地址，不含读写位
#define ZLG7290_I2C_ID   0x38
```

```
//定义 ZLG7290 内部寄存器地址（子地址）
#define ZLG7290_SystemReg       0x00    //系统寄存器
#define ZLG7290_Key             0x01    //键值寄存器
#define ZLG7290_RepeatCnt       0x02    //连击次数寄存器
#define ZLG7290_FunctionKey     0x03    //功能键寄存器
#define ZLG7290_CmdBuf          0x07    //命令缓冲区起始地址
#define ZLG7290_CmdBuf0         0x07    //命令缓冲区 0
#define ZLG7290_CmdBuf1         0x08    //命令缓冲区 1
#define ZLG7290_FlashOnOff      0x0C    //闪烁控制寄存器
#define ZLG7290_ScanNum         0x0D    //扫描位数寄存器
#define ZLG7290_DpRam           0x10    //显示缓存起始地址
#define ZLG7290_DpRam0          0x10    //显示缓存 0
#define ZLG7290_DpRam1          0x11    //显示缓存 1
#define ZLG7290_DpRam2          0x12    //显示缓存 2
#define ZLG7290_DpRam3          0x13    //显示缓存 3
#define ZLG7290_DpRam4          0x14    //显示缓存 4
#define ZLG7290_DpRam5          0x15    //显示缓存 5
#define ZLG7290_DpRam6          0x16    //显示缓存 6
#define ZLG7290_DpRam7          0x17    //显示缓存 7

//向 ZLG7290 的某个内部寄存器写入数据
bit ZLG7290_WriteReg(unsigned char RegAddr, char dat);

//从 ZLG7290 的某个内部寄存器读出数据
bit ZLG7290_ReadReg(unsigned char RegAddr, char *dat);

//向 ZLG7290 发送控制命令
bit ZLG7290_cmd(char cmd0, char cmd1);

//段寻址，单独点亮或熄灭数码管（或 LED）中的某一段
bit ZLG7290_SegOnOff(char seg, bit b);

//下载数据并译码
bit ZLG7290_Download(char addr, bit dp, bit flash, char dat);

//闪烁控制指令（Fn 应当是字节型）
//Fn 的 8 个位分别控制数码管的 8 个位是否闪烁，0－不闪烁，1－闪烁
#define ZLG7290_Flash(Fn)       ZLG7290_cmd(0x70,(Fn))

#endif   //_ZLG7290_H_
```

9.4 C 文件程序清单

程序清单 9.2 ZLG7290B 的 C51 驱动程序 C 文件

```
/*
	ZLG7290.c
	数码管显示与键盘管理芯片 ZLG7290 的标准 80C51 驱动程序 C 文件
	Copyright (c) 2005，广州周立功单片机发展有限公司
	All rights reserved.
	本程序仅供学习参考，不提供任何可靠性方面的担保；请勿用于商业目的
*/

#include "I2C.h"
#include "ZLG7290.h"

/*
函数：ZLG7290_WriteReg()
功能：向 ZLG7290 的某个内部寄存器写入数据
参数：
	RegAddr：ZLG7290 的内部寄存器地址
	dat：要写入的数据
返回：
	0：正常
	1：访问 ZLG7290 时出现异常
*/
bit ZLG7290_WriteReg(unsigned char RegAddr, char dat)
{
	bit b;
	b = I2C_Puts(ZLG7290_I2C_ID,RegAddr,1,&dat,1);
	return b;
}

/*
函数：ZLG7290_ReadReg()
功能：从 ZLG7290 的某个内部寄存器读出数据
参数：
	RegAddr：ZLG7290 的内部寄存器地址
	*dat：保存读出的数据
返回：
	0：正常
	1：访问 ZLG7290 时出现异常
*/
bit ZLG7290_ReadReg(unsigned char RegAddr, char *dat)
{
	bit b;
	b = I2C_Gets(ZLG7290_I2C_ID,RegAddr,1,dat,1);
	return b;
}
```

```
/*
函数：ZLG7290_cmd()
功能：向 ZLG7290 发送控制命令
参数：
      cmd0：写入 CmdBuf0 寄存器的命令字（第 1 字节）
      cmd1：写入 CmdBuf1 寄存器的命令字（第 2 字节）
返回：
      0：正常
      1：访问 ZLG7290 时出现异常
*/
bit ZLG7290_cmd(char cmd0, char cmd1)
{
      bit b;
      char buf[2];
      buf[0] = cmd0;
      buf[1] = cmd1;
      b = I2C_Puts(ZLG7290_I2C_ID,ZLG7290_CmdBuf,1,buf,2);
      return b;
}

/*
函数：ZLG7290_SegOnOff()
功能：段寻址，单独点亮或熄灭数码管（或 LED）中的某一段
参数：
      seg：取值 0～63，表示数码管（或 LED）的段号
      b：0 表示熄灭，1 表示点亮
返回：
      0：正常
      1：访问 ZLG7290 时出现异常
说明：
      在每一位数码管中，段号顺序按照“a,b,c,d,e,f,g,dp”进行
*/
bit ZLG7290_SegOnOff(char seg, bit b)
{
      char cmd;
      cmd = seg & 0x3F;
      if ( b ) cmd |= 0x80;
      return ZLG7290_cmd(0x01,cmd);
}

/*
函数：ZLG7290_Download()
功能：下载数据并译码
```

```
参数：
    addr：取值 0～7，显示缓存 DpRam0～DpRam7 的编号
    dp：是否点亮该位的小数点，0－熄灭，1－点亮
    flash：控制该位是否闪烁，0－不闪烁，1－闪烁
    dat：取值 0～31，表示要显示的数据
返回：
    0：正常
    1：访问 ZLG7290 时出现异常
说明：
    显示数据具体的译码方式请参见 ZLG7290 的数据手册
*/
bit ZLG7290_Download(char addr, bit dp, bit flash, char dat)
{
    char cmd0;
    char cmd1;
    cmd0 = addr & 0x0F;
    cmd0 |= 0x60;
    cmd1 = dat & 0x1F;
    if ( dp ) cmd1 |= 0x80;
    if ( flash ) cmd1 |= 0x40;
    return ZLG7290_cmd(cmd0,cmd1);
}
```

第10章 ZLG7290B 演示程序

10.1 演示程序说明

ZLG7290B 的 C51 演示程序“ZLG7290Demo.c”是以第 3 章的典型应用电路图（图 3.1）为基础的，首先详细演示了 ZLG7290B 的各项数码管显示功能，最后清除所有显示，并等待某个键按下并将按键值显示出来（同时包括 Key、RepeatCnt 和 FunctionKey 的值）。演示程序都将用到第 8 章和第 9 章中的 C51 驱动程序软件包。该演示程序已经在 ZLG7290B 的 Test 电路板上调试通过。

10.2 演示程序清单

程序清单 10.1 ZLG7290B 的演示程序

```
/*
    ZLG7290 演示程序
*/

#include "I2C.h"
#include "ZLG7290.h"

//定义键盘中断标志，FlagINT=1 表示有键按下
volatile bit FlagINT = 0;

/*
函数：INT0_SVC()
功能：ZLG7290 键盘中断服务程序
说明：中断触发方式选择负边沿触发，因此不必等待中断请求信号恢复为高电平
*/
void INT0_SVC() interrupt 0
{
    FlagINT = 1;
}

/*
函数：Delay()
功能：延时 10ms～655.36s
参数：
    t>0 时，延时(t*0.01)s
    t=0 时，延时 655.36s
说明：
    晶振采用 11.0592MHz
*/
void Delay(unsigned int t)
{
    do
```

```
    {
        TH0 = 0xDC;
        TL0 = 0x00;
        TR0 = 1;
        while ( !TF0 );
        TF0 = 0;
        TR0 = 0;
    } while (--t);
}

/*
函数：SystemInit()
功能：系统初始化
*/
void SystemInit()
{
    I2C_Init();
    TMOD = 0x01;
    Delay(30);        //等待 ZLG7290 复位完毕
}

/*
函数：ClearAll()
功能：清除所有显示
*/
void ClearAll()
{
    unsigned char x;
    for ( x=0; x<8; x++ )
    {
        ZLG7290_Download(x,0,0,31);
    }
}

/*
函数：Test_DispBuf()
功能：测试直接写显存
*/
void Test_DispBuf()
{
    code char DispDat[16] =
    {//字母 AbCdEFgHiJkLoPqr 的字形数据
        0xEE,0x3E,0x9C,0x7A,0x9E,0x8E,0xF6,0x6E,
        0x20,0x70,0x0E,0x1C,0x3A,0xCE,0xE6,0x0A
```

```
    };
    unsigned char n;
    unsigned char x;
    unsigned char reg;
    unsigned char dat;
    for ( n=0; n<16; n++ )
    {
        for ( x=0; x<8; x++ )
        {
            reg = ZLG7290_DpRam + x;
            dat = DispDat[n];
            ZLG7290_WriteReg(reg,dat);
        }
        Delay(50);
    }
}

/*
函数：Test_Download()
功能：测试下载数据功能
*/
void Test_Download()
{
    unsigned char x;
    bit dp;
    bit flash;
    char dat;
//点亮所有数码管
    dp = 1;
    flash = 0;
    dat = 8;
    for ( x=0; x<8; x++ )
    {
        ZLG7290_Download(x,dp,flash,dat);
    }
    Delay(100);
//依次显示所有字型
    dp = 0;
    flash = 0;
    for ( dat=0; dat<32; dat++ )
    {
        for ( x=0; x<8; x++ )
        {
            ZLG7290_Download(x,dp,flash,dat);
```

```
            }
            Delay(50);
        }
}

/*
函数：Test_ScanNum()
功能：测试不同扫描位数
说明：扫描位数越少，数码管就越亮
*/
void Test_ScanNum()
{
        unsigned char x;
        for ( x=0; x<8; x++ )
        {
            ZLG7290_Download(x,1,0,8);
        }
        Delay(100);
        for ( x=0; x<8; x++ )
        {
            ZLG7290_WriteReg(ZLG7290_ScanNum,x);
            Delay(100);
        }
}

/*
函数：Test_Flash()
功能：测试闪烁功能
*/
void Test_Flash()
{
        char dat = 0x01;
        unsigned char x;
//显示 01234567
        for ( x=0; x<8; x++ )
        {
            ZLG7290_Download(x,0,0,x);
        }
//设置闪烁控制寄存器
        ZLG7290_WriteReg(ZLG7290_FlashOnOff,0x11);
//闪烁演示
        for ( x=0; x<8; x++ )
        {
            ZLG7290_Flash(dat);
```

```
            dat <<= 1;
            Delay(300);
        }
//数码管的 8 个位一起闪烁
        ZLG7290_Flash(0xFF);
        Delay(350);
//停止闪烁
        ZLG7290_Flash(0x00);
        Delay(50);
}

/*
函数：Test_SegOnOff()
功能：测试段寻址功能
*/
void Test_SegOnOff()
{
        unsigned char seg;
        ClearAll();
        Delay(100);
        for ( seg=0; seg<64; seg++ )
        {
            ZLG7290_SegOnOff(seg,1);
            Delay(30);
        }
        Delay(100);
        for ( seg=0; seg<64; seg++ )
        {
            ZLG7290_SegOnOff(seg,0);
            Delay(30);
        }
        Delay(100);
}

/*
函数：DispValue()
功能：显示 100 以内的数值
参数：
        x：显示位置，取值 0～6
        dat：要显示的数据，取值 0～99
*/
void DispValue(char x, unsigned char dat)
{
        unsigned char d;
```

```
    d = dat / 10;
    ZLG7290_Download(x,0,0,d);
    d = dat - d * 10;
    ZLG7290_Download(x+1,0,0,d);
}

/*
函数：DispHexValue()
功能：以 16 进制方式显示数值
参数：
    x：显示位置，取值 0～6
    dat：要显示的数据，取值 0～255
*/
void DispHexValue(char x, unsigned char dat)
{
    unsigned char d;
    d = dat / 16;
    ZLG7290_Download(x,0,0,d);
    d = dat - d * 16;
    ZLG7290_Download(x+1,0,0,d);
}

/*
函数：Test_Key()
功能：测试按键功能
*/
void Test_Key()
{
    unsigned char KeyValue;
    unsigned char RepeatCnt;
    unsigned char FnKeyValue;
    ClearAll();
    EA = 0;
    IT0 = 1;    //负边沿触发中断
    EX0 = 1;    //允许外部中断
    EA = 1;
    for (;;)
    {
        if ( FlagINT )          //如果有键按下
        {
        //清除中断标志
            FlagINT = 0;
        //读取键值、连击计数器值、功能键值
            ZLG7290_ReadReg(ZLG7290_Key,&KeyValue);
```

```
                ZLG7290_ReadReg(ZLG7290_RepeatCnt,&RepeatCnt);
                ZLG7290_ReadReg(ZLG7290_FunctionKey,&FnKeyValue);
            //显示键值、连击计数器值、功能键值
                DispValue(0,KeyValue);
                DispHexValue(3,RepeatCnt);
                DispHexValue(6,FnKeyValue);
            }
            PCON |= 0x01;  //使 CPU 进入空闲状态，任一中断可唤醒
        }
}

void main()
{
        SystemInit();           //系统初始化
        Test_DispBuf();         //测试直接写显存
        Test_Download();        //测试下载数据
        Test_ScanNum();         //测试不同扫描位数
        Test_Flash();           //测试闪烁功能
        Test_SegOnOff();        //测试段寻址功能
        Test_Key();             //测试键盘功能
        while (1);
}
```

第11章 参考文献

[1]周立功单片机. 《ZLG7290 I^2C 接口键盘及 LED 驱动器数据手册》.http://www.zlgmcu.com

[2]马忠梅,戚军,刘滨,马岩.《单片机 C 语言 Windows 环境编程宝典》.北京航空航天大学出版社.2003

[3]何立民. 《I^2C 总线应用系统设计》.北京航空航天大学出版社.1995