

液 晶 LCD1602

(中文资料)

<http://hi.baidu.com/ball648500361>

ball

2010-5-13 整理

目录

1. 指令介绍
2. 显示字符
3. 显示自定义字符
4. 使用 4 线数据传输

1. 指令介绍

LCD1602 已很普遍了，具体介绍我就不多说了，市面上字符液晶绝大多数是基于 HD44780 液晶芯片的，控制原理是完全相同的，因此 HD44780 写的控制程序可以很方便地应用于市面上大部分的字符型液晶。字符型 LCD 通常有 14 条引脚线或 16 条引脚线的 LCD，多出来的 2 条线是背光电源线 VCC(15 脚)和地线 GND(16 脚)，其控制原理与 14 脚的 LCD 完全一样，定义如下表所示：

字符型 LCD 的引脚定义

引脚号	引脚名	电平	输入/输出	作用
1	Vss			电源地
2	Vcc			电源(+5V)
3	Vee			对比调整电压
4	RS	0/1	输入	0=输入指令 1=输入数据
5	R/W	0/1	输入	0=向LCD写入指令或数据 1=从LCD读取信息
6	E	1, 1→0	输入	使能信号，1时读取信息， 1→0(下降沿)执行指令
7	DB0	0/1	输入/输出	数据总线line0(最低位)
8	DB1	0/1	输入/输出	数据总线line1
9	DB2	0/1	输入/输出	数据总线line2
10	DB3	0/1	输入/输出	数据总线line3
11	DB4	0/1	输入/输出	数据总线line4
12	DB5	0/1	输入/输出	数据总线line5
13	DB6	0/1	输入/输出	数据总线line6
14	DB7	0/1	输入/输出	数据总线line7(最高位)
15	A	+Vcc		LCD背光电源正极
16	K	接地		LCD背光电源负极

HD44780 内置了 DDRAM, CGROM 和 CGRAM

这里的 RAM 与 ROM 不懂得话, 可以看看我整理的另一篇

<http://hi.baidu.com/ball648500361/blog/item/4332fdf8bf505fd2b48f3150.html>

DDRAM 就是显示数据 RAM 用来寄存待显示的字符代码。共 80 个字节, 其地址和屏幕的对应关系如下表:

	显示位置	1	2	3	4	5	6	7	40
DDRAM	第一行	00H	01H	02H	03H	04H	05H	06H	27H
地 址	第二行	40H	41H	42H	43H	44H	45H	46H	67H

也就是说想要在 LCD1602 屏幕的第一行第一列显示一个“A”字, 就要向 DDRAM 的 00H 地址写入“A”字的代码就行了。但具体的写入是要按 LCD 模块的指令格式来进行的, 后面我会说到的。那么一行可有 40 个地址呀? 是的, 在 1602 中我们就用前 16 个就行了。第二行也一样用前 16 个地址。对应如下:

DDRAM 地址与显示位置的对应关系

00H	01H	02H	03H	04H	05H	06H	07H	08H	09H	0AH	0BH	0CH	0DH	0EH	0FH
40H	41H	42H	43H	44H	45H	46H	47H	48H	49H	4AH	4BH	4CH	4DH	4EH	4FH

我们知道文本文件中每一个字符都是用一个字节的代码记录的。一个汉字是用两个字节的代码记录。在 PC 上我们只要打开文本文件就能在屏幕上看到对应的字符是因为在操作系统里和 BIOS 里都固化有字符字模。什么是字模? 就代表了是在点阵屏幕上点亮和熄灭的信息数据。例如“A”

字的字模:

01110	○ ■ ■ ■ ○
10001	■ ○ ○ ○ ■
10001	■ ○ ○ ○ ■
10001	■ ○ ○ ○ ■
11111	■ ■ ■ ■ ■

10001 ■ ○ ○ ○ ■

10001 ■ ○ ○ ○ ■

上图左边的数据就是字模数据，右边就是将左边数据用“○”代表0，用“■”代表1。看出是个“A”字了吗？在文本文件中“A”字的代码是41H，PC收到41H的代码后就去字模文件中将代表A字的这一组数据送到显卡去点亮屏幕上相应的点，你就看到“A”这个字了。

刚才我说了想要在 LCD1602 屏幕的第一行第一列显示一个“A”字，就要向 DDRAM 的 00H 地址写入“A”字的代码 41H 就行了，可 41H 这一个字节的代码如何才能让 LCD 模块在屏幕的阵点上显示“A”字呢？同样，在 LCD 模块上也固化了字模存储器，这就是 CGROM 和 CGRAM。HD44780 内置了 192 个常用字符的字模，存于字符产生器 CGROM (Character Generator ROM) 中，另外还有 8 个允许用户自定义的字符产生 RAM，称为 CGRAM (Character Generator RAM)。下图说明了 CGROM 和 CGRAM 与字符的对应关系。

CGROM 中字符码与字模关系对照表

↓	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	
xxxx0000	CG RAM (1)			0	a	P	`	P				一	夕	ミ	α	p	
xxxx0001	(2)			!	1	A	Q	a	4			。	ア	チ	△	ä	q
xxxx0010	(3)			"	2	B	R	b	r			「	イ	ツ	×	β	θ
xxxx0011	(4)			#	3	C	S	c	s			」	ウ	テ	モ	ε	∞
xxxx0100	(5)			\$	4	D	T	d	t			、	エ	ト	ヤ	μ	Ω
xxxx0101	(6)			%	5	E	U	e	u			・	オ	ナ	1	ε	Ü
xxxx0110	(7)			&	6	F	V	f	v			ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)			'	7	G	W	g	w			ア	キ	ヌ	ラ	q	π
xxxx1000	(1)			<	8	H	X	h	x			ィ	ク	ネ	リ	フ	×
xxxx1001	(2)			>	9	I	Y	i	y			ッ	ケ	ル	ル	”	γ
xxxx1010	(3)			*	:	J	Z	j	z			エ	コ	ハ	レ	j	〒
xxxx1011	(4)			+	;	K	[k	[オ	サ	ヒ	ロ	*	斤
xxxx1100	(5)			,	<	L	¥	l	l			ハ	シ	フ	ワ	¢	円
xxxx1101	(6)			-	=	M]	m]			ユ	ズ	ヘ	ン	も	÷
xxxx1110	(7)			.	>	N	^	n	+			ヨ	セ	ホ	”	ん	
xxxx1111	(8)			/	?	O	_	o	+			ッ	ソ	マ	”	〇	■

从上图可以看出,“A”字的对应上面高位代码为 0100,对应左边低位代码为 0001,合起来就是 01000001,也就是 41H。可见它的代码与我们 PC 中的字符代码是基本一致的。因此我们在向 DDRAM 写 C51 字符代码程序时甚至可以直接用 P1 = 'A' 这样的方法。PC 在编译时就把“A”先转为 41H 代码了。

字符代码 0x00 ~ 0x0F 为用户自定义的字符图形 RAM(对于 5X8 点阵的字符,可以存放 8 组,5X10 点阵的字符,存放 4 组),就是 CGRAM 了。后面我会详细说的。

0x20 ~ 0x7F 为标准的 ASCII 码,0xA0 ~ 0xFF 为日文字符和希腊文字符,其余字符码(0x10 ~ 0x1F 及 0x80 ~ 0x9F)没有定义。

那么如何对 DDRAM 的内容和地址进行具体操作呢,下面先说说 HD44780 的指令集及其设置说明,请浏览该指令集,并找出对 DDRAM 的内容和地址进行操作的指令。

共 11 条指令:

1. 清屏指令

指令功能	指令编码										执行时间 /ms
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
清屏	0	0	0	0	0	0	0	0	0	1	1.64

功能: <1> 清除液晶显示器,即将 DDRAM 的内容全部填入"空白"的 ASCII 码 20H;

<2> 光标归位,即将光标撤回液晶显示屏的左上方;

<3> 将地址计数器(AC)的值设为 0。

2. 光标归位指令

指令功能	指令编码										执行时间 /ms
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
光标归位	0	0	0	0	0	0	0	0	1	X	1.64

功能: <1> 把光标撤回回到显示器的左上方;

<2> 把地址计数器(AC)的值设置为 0;

<3> 保持 DDRAM 的内容不变。

3. 进入模式设置指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
进入模式设置	0	0	0	0	0	0	0	1	I/D	S	40

功能: 设定每次定入 1 位数据后光标的移位方向, 并且设定每次写入的一个字符是否移动。参数设定的

情况如下所示:

位名 设置

I/D 0=写入新数据后光标左移 1=写入新数据后光标右移

S 0=写入新数据后显示屏不移动 1=写入新数据后显示屏整体右移 1 个字符

4. 显示开关控制指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
显示开关控制	0	0	0	0	0	0	1	D	C	B	40

功能: 控制显示器开/关、光标显示/关闭以及光标是否闪烁。参数设定的情况如下:

位名 设置

D 0=显示功能关 1=显示功能开

C 0=无光标 1=有光标

B 0=光标闪烁 1=光标不闪烁

5. 设定显示屏或光标移动方向指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
设定显示屏或光标移动方向	0	0	0	0	0	1	S/C	R/L	X	X	40

功能：使光标移位或使整个显示屏幕移位。参数设定的情况如下：

S/C	R/L	设定情况
0	0	光标左移 1 格，且 AC 值减 1
0	1	光标右移 1 格，且 AC 值加 1
1	0	显示器上字符全部左移一格，但光标不动
1	1	显示器上字符全部右移一格，但光标不动

6. 功能设定指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
功能设定	0	0	0	0	1	DL	N	F	X	X	40

功能：设定数据总线位数、显示的行数及字型。参数设定的情况如下：

位名	设置
DL	0=数据总线为 4 位 1=数据总线为 8 位
N	0=显示 1 行 1=显示 2 行
F	0=5×7 点阵/每字符 1=5×10 点阵/每字符

7. 设定 CGRAM 地址指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
设定 CGRAM 地址	0	0	0	1	CGRAM 的地址 (6 位)						40

功能：设定下一个要存入数据的 CGRAM 的地址。

DB5DB4DB3 为字符号，也就是你将来要显示该字符时要用到的字符地址。(000~111) (能定义八个字符)

DB2DB1DB0 为行号。(000~111) (八行)

8. 设定 DDRAM 地址指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
设定 DDRAM 地址	0	0	1	CGRAM的地址(7位)							40

功能：设定下一个要存入数据的 DDRAM 的地址。

9. 读取忙碌信号或 AC 地址指令

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
读取忙碌信号或 AC 地址	0	1	FB	AC 内容(7位)							40

功能：<1> 读取忙碌信号 BF 的内容，BF=1 表示液晶显示器忙，暂时无法接收单片机送来的数据或指令；

当 BF=0 时，液晶显示器可以接收单片机送来的数据或指令；

<2> 读取地址计数器(AC)的内容。

10. 数据写入 DDRAM 或 CGRAM 指令一览

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
数据写入到 DDRAM 或 CGRAM	1	0	要写入的数据 D7~D0								40

功能：<1> 将字符码写入 DDRAM 以使液晶显示屏显示出相对应的字符；

<2> 将使用者自己设计的图形存入 CGRAM

DB7DB6DB5 可为任何数据，一般取“000”。

DB4DB3DB2DB1DB0 对应于每行 5 点的字模数据。

11. 从 CGRAM 或 DDRAM 读出数据的指令一览

指令功能	指令编码										执行时间/ μ s
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
从CGRAM或DDRAM读出数据	1	1	要读出的数据 D7~D0								40

功能：读取 DDRAM 或 CGRAM 中的内容。

基本操作时序：

读状态 输入：RS=L, RW=H, E=H 输出：DB0 ~ DB7=状态字

写指令 输入：RS=L, RW=L, E=下降沿脉冲, DB0 ~ DB7=指令码 输出：无

读数据 输入：RS=H, RW=H, E=H 输出：DB0 ~ DB7=数据

写数据 输入：RS=H, RW=L, E=下降沿脉冲, DB0 ~ DB7=数据 输出：无

2. 显示字符

看了那么多是不是有些晕？我也是啊，不过慢慢理解还是没问题的。

实际上面说了那么多具体怎么操作我还是没会啊？好！咱就简单点。

举个实例，就在 LCD1602 屏幕上第一行第一列显示个“A”字。

1. 先初始化。（老大！好像上面没初始化这条指令啊！）

先别拿东西扔我，说明书上是这么说的。也就先写入些指令。

//先定义接口

```
# include <AT89x51.h>
/*****
P1-----DB0 ~ DB7          P2. 0-----RS
P2. 1-----RW
P2. 2-----E
*****/
# define LCD_DB      P1
sbit      LCD_RS=P2^0;
sbit      LCD_RW=P2^1;
sbit      LCD_E=P2^2;
/*****定义函数*****/
# define uchar unsigned char
# define uint unsigned int
```

```

void LCD_init(void); //初始化函数
void LCD_write_command(uchar command); //写指令函数
void LCD_write_data(uchar dat); //写数据函数
void LCD_disp_char(uchar x, uchar y, uchar dat); //在某个屏幕位置上显示一个字符, X (0-16), y(1-2)
//void LCD_check_busy(void); //检查忙函数。我没用到此函数, 因为通过率极低。
void delay_n40us(uint n); //延时函数
//*****
//*****初始化函数*****
void LCD_init(void)
{
    LCD_write_command(0x38); //设置 8 位格式, 2 行, 5x7
    LCD_write_command(0x0c); //整体显示, 关光标, 不闪烁
    LCD_write_command(0x06); //设定输入方式, 增量不移位
    LCD_write_command(0x01); //清除屏幕显示
    delay_n40us(100); //实践证明, 我的 LCD1602 上, 用 for 循环 200 次就能可靠完成清屏指令。
}
//*****
//*****写指令函数*****
void LCD_write_command(uchar dat)
{
    LCD_DB=dat;
    LCD_RS=0; //指令
    LCD_RW=0; //写入
    LCD_E=1; //允许
    LCD_E=0;
    delay_n40us(1); //实践证明, 我的 LCD1602 上, 用 for 循环 1 次就能完成普通写指令。
}
//*****
//*****写数据函数*****
void LCD_write_data(uchar dat)
{
    LCD_DB=dat;
    LCD_RS=1; //数据
    LCD_RW=0; //写入
    LCD_E=1; //允许
    LCD_E=0;
    delay_n40us(1);
}
//*****
//*****显示一个字符函数*****

```

```

void LCD_disp_char(uchar x, uchar y, uchar dat)
{
    uchar address;
    if(y==1)
        address=0x80+x;
    else
        address=0xc0+x;
    LCD_write_command(address);
    LCD_write_data(dat);
}
//*****
//*****检查忙函数*****
void LCD_check_busy()    //实践证明，在我的 LCD1602 上，检查忙指令通
                        //过率极低，以
                        //至于不能正常使用 LCD。因
{
    //此我没有再用检查忙函数。而使
    do
        //用了延时的方法，延时还是非
        //常好用的。我试了一下，用
        { LCD_E=0;
            //for 循环作延时，普通指令只要 1
            次循环就可完成。清屏指令
            LCD_RS=0;
            //要用 200 次循环便能完成。
            LCD_RW=1;
            LCD_DB=0xff;
            LCD_E=1;
        }while(LCD_DB^7==1);
}
//*****
//*****延时函数*****
void delay_n40us(uint n)
{
    uint i;
    uchar j;
    for(i=n; i>0; i--)
        for(j=0; j<2; j++);
    //在这个延时循环函数中我只做了
    2 次循环，
}
//实践证明我的 LCD1602 上普
通的指令只需 1 次循环就能可靠完成。
//*****
//*****主函数*****
void main(void)
{
    LCD_init();
    LCD_disp_char(0, 1, "A");
    while(1);
}

```

//*****

3. 显示自定义字符

上面只是显示了‘A’这一个字符（即将 CGROM 中的‘A’写入 DDRAM）；

下面我再补充一下：（显示一个自定义字符）

步骤如下：

1. 先将自定义字符写入 CGRAM
2. 再将 CGRAM 中的自定义字符送到 DDRAM 中显示

很简单的：看好了

查看 LCD1602 的 CGROM 字符代码表，可以发现从 00000000B~00000111B（00H~07H）地址的内容是没有定义的，它是留给用户自己定义的，用户可以通过先定义 LCD1602 的 CGRAM 中的内容，然后就可以同调用 CGROM 字符一样来调用自定义好的字符（这里提示一下，自定义的字符最多可写 8 个）

那么如何设定 CGRAM 中的内容呢？首先我们要把所要编写的字符对应于 5X8 点阵的“字模”提取出来，我们可以通过相关的软件来提取，也可以手工提取。说白了也就是将点阵的某一行中有显示的点用 1 表示，无显示的点用 0 表示，以此形成该行对应的字模数据。

设定 CGRAM 的内容，要一行一行的设定，每一行对应一个 CGRAM 5X8 点阵，每行 5 点，共 8 行，因此要将 8 行的字模数据都写入 CGRAM。写好后，就可像调用 CGROM 字符一样来调用它了。

定义一行的内容，分两步：

1. 设定行地址（CGRAM 地址）：

用到的命令如下：

```
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
0 0 0 1 DATA
```

其中：DB5DB4DB3 为字符地址，也就是你将来要显示该字符时要用到的字符地址。
DB2DB1DB0 为行号。

2. 设定 CGRAM 数据（内容）指令码如下：

```
RS R/W DB7 DB6 DB5 DB4 DB3 DB2 DB1 DB0
1 0 DATA
```

其中：DB4DB3DB2DB1DB0 对应于每行 5 点的字模数据。

DB7DB6DB5 可为任何数据，一般取“000”。

```
#include<reg52.h>
#include <intrins.h>
sbit lcd1602_rs=P0^7;
sbit lcd1602_rw=P0^6;
sbit lcd1602_e=P0^5;
#define lcd1602_sent_dat P2

#define lcd1602_dat lcd1602_rs=1;
```

```

#define lcd1602_com lcd1602_rs=0;
#define lcd1602_write lcd1602_rw=0;
#define lcd1602_read lcd1602_rw=1;
#define lcd1602_e_able lcd1602_e=1;
#define lcd1602_e_unable lcd1602_e=0;

//此函数定义八个字符分别写入 CGRAM 的八个地址
unsigned char code pic[8][8]={
/*-- 调入了四幅图像: 向上 --*/
/*-- 宽度 x 高度=5x8 --*/
/*-- 宽度不是 8 的倍数, 现调整为: 宽度 x 高度=8x8 --*/
{0x04, 0x0E, 0x15, 0x04, 0x04, 0x04, 0x04, 0x00}, // ↑
{0x00, 0x04, 0x04, 0x04, 0x04, 0x15, 0x0E, 0x04}, // ↓
{0x00, 0x04, 0x08, 0x1F, 0x08, 0x04, 0x00, 0x00}, // ←
{0x00, 0x04, 0x02, 0x1F, 0x02, 0x04, 0x00, 0x00}, // →
{0x04, 0x04, 0x0A, 0x1F, 0x1F, 0x0A, 0x04, 0x04}, // 占位符
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0},
{0, 0, 0, 0, 0, 0, 0, 0}
};

unsigned char lcd1602_read_dat()
{
    unsigned char dat;
    lcd1602_read;
    lcd1602_dat;
    lcd1602_e_able;
    dat=lcd1602_sent_dat;
    lcd1602_e_unable;
    return dat;
}

void read_busy()
{
    while(lcd1602_read_dat() & 0x80);
}

void lcd1602_write_dat(unsigned char dat)
{
    char i;
    lcd1602_sent_dat=dat;
    lcd1602_dat;
    lcd1602_write;
    for(i=0; i<80; i++)

```

```

        _nop_();
        lcd1602_e_able;
        for(i=0;i<80;i++)
            _nop_();
        lcd1602_e_unable;
        read_busy();
    }
void lcd1602_write_com(unsigned char com)
{
    char i;
    lcd1602_sent_dat=com;
    lcd1602_com;
    lcd1602_write;
    for(i=0;i<80;i++)
        _nop_();
    lcd1602_e_able;
    for(i=0;i<80;i++)
        _nop_();
    lcd1602_e_unable;
    read_busy();
}

//初始化
void init()
{
    read_busy();
    lcd1602_write_com(0x38);
    lcd1602_write_com(0x0c);
    lcd1602_write_com(0x06);
    lcd1602_write_com(0x01);
    lcd1602_write_com(0x02);
}

//显示字符串
void lcd1602_write_character(unsigned char add,unsigned char *p)
{
    lcd1602_write_com(add);
    while (*p!='\0')
    {
        lcd1602_write_dat(*p++);
    }
}

//将自定义字符写入 CGRAM

```

```

//add 是 CGRAM 的地址 (0~7)
/*pic_num是指向一个 8 位数组的首地址
void lcd1602_write_pic(unsigned char add,unsigned char *pic_num)
{
    unsigned char i;
    add=add<<3;
    for(i=0;i<8;i++)
    {
        lcd1602_write_com(0x40|add+i);
        lcd1602_write_dat(*pic_num++);
    }
}

void main()
{
    int i=0;
    init();
    //显示 CGROM中的字符
    lcd1602_write_com(0x80);
    lcd1602_write_dat('6');
    //写八个自定义字符到 CGRAM
    for(i=0;i<8;i++)
        lcd1602_write_pic(i,pic[i]);

    //显示八个自定义字符
    lcd1602_write_com(0x80+0X01);
    for(i=0;i<0x08;i++)
        lcd1602_write_dat(i);

    lcd1602_write_character(0x80+0x40,"test!success");
    while(1);
}

```

5. 使用 4 线数据传输

画完自定义字符这是第三步，在液晶屏刷新率不高的情况下，我们可以用 1602 的另一种 4 线数据传输可减少 io 口的使用数量
希望大家喜欢，由于没有找到相关资料这里只将程序贴出来给大家参考参考

```

#include<reg52.h>
#include<intrins.h>
sbit lcd1602_rs=P0^7;
sbit lcd1602_rw=P0^6;
sbit lcd1602_e=P0^5;
/*#define lcd1602_io_writ lcd1602_rw=0;
#define lcd1602_io_read lcd1602_rw=1;

```



```

#define lcd1602_io_com lcd1602_rs=0;
#define lcd1602_io_dat lcd1602_rs=1;
#define lcd1602_io_able lcd1602_e=0;
#define lcd1602_io_unable lcd1602_e=1;
*/
sbit lcd1602_dat1=P0^0;
sbit lcd1602_dat2=P0^1;
sbit lcd1602_dat3=P0^2;
sbit lcd1602_dat4=P0^3;
//unsigned char code table[]="ball hahhahhaha";
//unsigned char code table1[]="THE BEST!";
void delay(unsigned int z)
{
    unsigned int a,b;
    for(a=z;a>0;a--)
        for(b=114;b>0;b--);
}
void lcd1602_write(bit cd,unsigned char dat)//cd=0,指令; cd=1,数据
{
    int i;
    unsigned char j;
    lcd1602_rs=cd;
    for(j=0;j<2;j++)
    {
        lcd1602_e=1;
        for(i=0;i<200;i++)
            _nop_();
        lcd1602_dat4=dat&0x80;
        lcd1602_dat3=(dat<<1)&0x80;
        lcd1602_dat2=(dat<<2)&0x80;
        lcd1602_dat1=(dat<<3)&0x80;
        lcd1602_e=0;
        //lcd1602_e=1;
        dat<<=4;
    }
}

/*unsigned char lcd1602_read_dat(unsigned char add)
{
    unsigned char dat;
    return dat;
}*/
void main()

```

```
{  
lcd1602_rw=0;  
lcd1602_e=0;  
lcd1602_write(0, 0x28);  
lcd1602_write(0, 0x0f);  
lcd1602_write(0, 0x06);  
lcd1602_write(0, 0x01);  
lcd1602_write(0, 0x80);  
lcd1602_write(1, 'q');  
while(1);  
}
```

声明：资料大部分是由网上搜索而来，如有侵权请联系作者 ball

QQ648500361

百度空间

<http://hi.baidu.com/ball648500361>